▼ AMSTRAD ▼ SPECTRUM ▼ COMMODORE ▼ IBM ▼ MSX ▼

Informática y programación 15

PASO A PASO



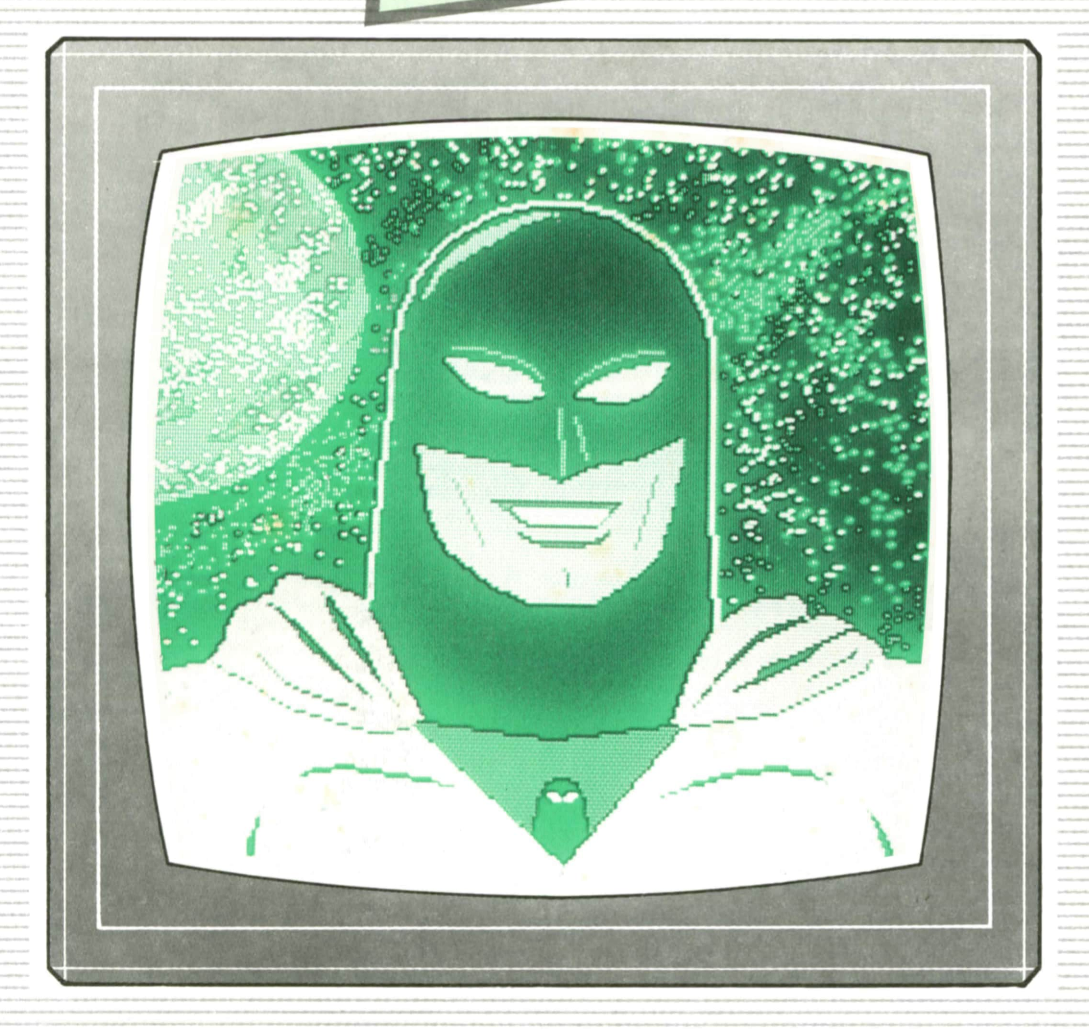
PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Informática y programación 15

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Una publicación de

EDICIONES SIGLO CULTURAL, S.A.

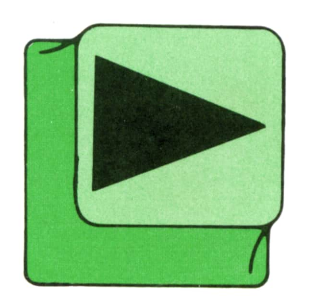
Director-editor:
RICARDO ESPAÑOL CRESPO.
Gerente:
ANTONIO G. CUERPO.
Directora de producción:
MARIA LUISA SUAREZ PEREZ.
Directores de la colección:
MANUEL ALFONSECA, Doctor Ingeniero de Telecomunicación y Licenciado en Informática.
JOSE ARTECHE, Ingeniero de Telecomunicación.
Diseño y maquetación:
BRAVO-LOFISH.
Fotografía:
EQUIPO GALATA.
Dibujos:
JOSE OCHOA

TECNICAS DE PROGRAMACION: Manuel Alfonseca, Doctor Ingeniero de Telecomunicación y Licenciado en Informática. TECNICAS DE ANALISIS: José Arteche, Ingeniero en Telecomunicación. LENGUAJE MAQUINA 8086: Juan Rojas Licenciado en Ciencias Físicas e Ingeniero Industrial. PASCAL: Juan Ignacio Puyol, Ingeniero Industrial. PROGRAMAS (educativos, de utilidad, de gestión y de juegos): Francisco Morales, Técnico en Informática y colaboradores. Coordinador de AULA DE INFORMATICA APLICADA (AIA): Alejandro Marcos, Licenciado en Ciencias Químicas. BASIC: Esther Maldonado, Diplomada en Arquitectura. INFORMATICA BASICA: Virginia Muñoz, Diplomada en Informática. LENGUAJE MAQUINA Z-80: Joaquín Salvachúa, Diplomado en Telecomunicación y José Luis Tojo, Diplomado en Telecomunicación. LENGUAJE MAQUINA 6502: (desde el tomo 5): Juan José Gómez, Licenciado en Química. LOGO: Cristina Manzanera, Licenciada en Informática. APLICACIONES: Sociedad Tamariz, Diplomada en Telecomunicación. OTROS LENGUAJES (COBOL): Eloy Pérez, Licenciado en Informática. Ana Pastor, Licenciada en Informática.

Ediciones Siglo Cultural, S.A.
Dirección, redacción y administración:
Pedro Teixeira, 8, 2.ª planta. Teléf. 810 52 13. 28020 Madrid.
Publicidad:
Gofar Publicidad, S.A. Benito de Castro, 12 bis. 28028 Madrid.
Distribución en España:
COEDIS, S.A. Valencia, 245. Teléf. 215 70 97. 08007 Barcelona.
Delegación en Madrid: Serrano, 165. Teléf. 411 11 48.
Distribución en Ecuador: Muñoz Hnos.
Distribución en Perú: DISELPESA.
Distribución en Chile: Alfa Ltda.
Importador exclusivo Cono Sur:
CADE, S.R.L. Pasaje Sud América, 1532. Teléf.: 21 24 64.
Buenos Aires - 1.290. Argentina.

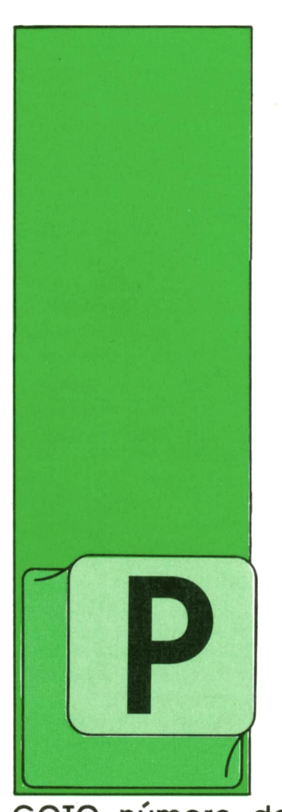
ISBN del tomo: 84-7688-107-X
ISBN de la obra: 84-7688-068-7
Fotocomposición:
ARTECOMP, S.A. Albarracín, 50. 28037 Madrid.
Imprime:
MATEU CROMO. Pinto (Madrid).

Depósito legal: M-5-677-1987
Printed in Spain - Impreso en España.
Suscripciones y números atrasados:
Ediciones Siglo Cultural, S.A.
Pedro Teixeira, 8, 2.ª planta. Teléf. 810 52 13. 28020 Madrid.
Junio, 1987.
P.V.P. Canarias: 335,–.



INDICE

BASIC





La instrucción ON-GOTO

PODEMOS conseguir también transferencias de control condicionales múltiples mediante la instrucción ON-GOTO, cuyo formato es el siguiente:

ON <expresión> GOTO número de línea, número de línea,... donde **expresión** es una variable numérica o una expresión aritmética.

Al encontrar está instrucción el ordenador evalúa la expresión y dirige el control a una línea u otra.

Por ejemplo:

50 ON DATO GOTO 200,400,600

— Si DATO=1... El control pasa a la línea 200

— Si DATO=2... El control pasa a la línea 400

— Si DATO=3... El control pasa a la línea 600

Por tanto, el efecto de la línea 50 es igual al que producirían la tres líneas siguientes:

50 IF DATO=1 THEN GOTO 200
60 IF DATO=2 THEN GOTO 400
70 IF DATO=3 THEN GOTO 600

Si la expresión no fuese un número entero, el ordenador lo truncaría, tomando la parte entera. Además, si el valor de la expresión fuese cero o mayor que el número de líneas especificadas, el control pasaría al primer número de línea indicado, aunque en algunos ordenadores la ejecución continuaría por la línea siguiente a la de la instrucción ON-GOTO.

El SPECTRUM es el único ordenador que no dispone de ON-GOTO; sin embargo, admite el siguiente formato para GOTO:

GOTO <expresión>

con lo cual el ejemplo anterior podríamos escribirlo así:

50 GOTO DATO*200

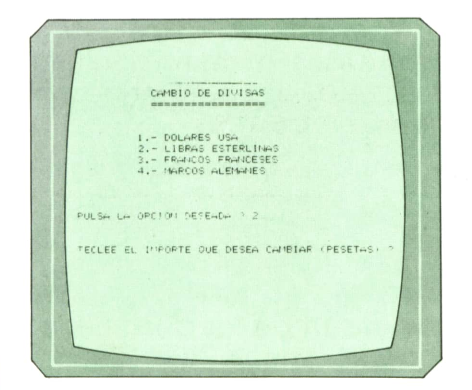
de modo que según DATO valga 1, 2, 3, etcétera, el control pasará a la línea 200, 400, 600, etc., respectivamente.

Veamos un ejemplo concreto. El programa permite calcular la cantidad exacta de dólares, libras esterlinas, francos franceses o marcos alemanes que obtendremos al cambiar por la cantidad deseada en pesetas.

```
10 REM ***********************
20 REM *  CAMBIO DE DIVISAS   *
30 REM ***********************
40 CLS
50 LET CDU=130:LET CLE=205:LET CFF=21:LET CMA=70
60 PRINT TAB(12);"CAMBIO DE DIVISAS"
70 PRINT TAB(12);"================="
80 PRINT :PRINT
90 PRINT TAB(10);"1.- DOLARES USA"
100 PRINT TAB(10);"2.- LIBRAS ESTERLINAS"
110 PRINT TAB(10);"3.- FRANCOS FRANCESES"
120 PRINT TAB(10);"4.- MARCOS ALEMANES"
130 PRINT :PRINT :PRINT
140 INPUT "PULSA LA OPCION DESEADA ";R
150 IF R<1 OR R>4 THEN GOTO 140
```

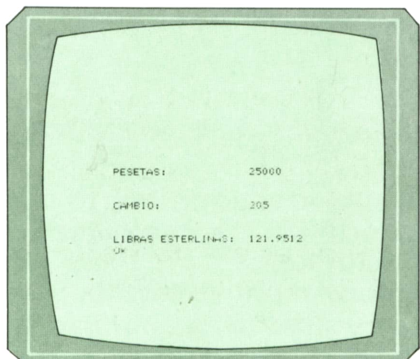
```
160 PRINT :PRINT
170 INPUT "TECLEE EL IMPORTE QUE DESEA CAMBIAR (PESETAS) ";P
180 CLS
190 PRINT "PESETAS:";TAB(20);P
200 PRINT :PRINT
210 ON R GOTO 300,400,500,600
300 REM * DOLARES USA *
310 LET DU=P/CDU
320 PRINT "CAMBIO:";TAB(20);CDU
330 PRINT :PRINT
340 PRINT "DOLARES USA:";TAB(20);DU
350 END
400 REM * LIBRAS ESTERLINAS *
410 LET LE=P/CLE
420 PRINT "CAMBIO:";TAB(20);CLE
430 PRINT :PRINT
440 PRINT "LIBRAS ESTERLINAS:";TAB(20);LE
450 END
500 REM * FRANCOS FRANCESES *
510 LET FF=P/CFF
520 PRINT "CAMBIO:";TAB(20);CFF
530 PRINT :PRINT
540 PRINT "FRANCOS FRANCESES:";TAB(20);FF
550 END
600 REM * MARCOS ALEMANES *
610 LET MA=P/CMA
620 PRINT "CAMBIO:";TAB(20);CMA
630 PRINT :PRINT
640 PRINT "MARCOS ALEMANES:";TAB(20);MA
```

En la línea 50 establecemos las cotizaciones de cambio de las cuatro monedas. De las líneas 70 a 170 se desarrolla una presentación en pantalla ordenada que nos permite seleccionar la opción de cambio (línea 140) e introducir la cantidad que deseamos cambiar (línea 170). El aspecto de la pantalla será algo similar al reproducido en la figura 6.



Presentación en pantalla del programa anterior.

En la línea 210 se transfiere el control a la línea correspondiente, según la opción elegida, donde se calcula el cambio y se imprimen los resultados. En la figura siguiente podemos ver la presentación final en pantalla tras una posible ejecución.



Aspecto de la pantalla tras una posible ejecución del programa.

Las presentaciones en pantalla están pensadas para un ordenador de 40 columnas, por tanto, si disponemos de una pantalla de 32 columnas como la del SPECTRUM, tendremos que cambiar todos los TAB(10) que aparecen por TAB(5) (y los TAB(12) por TAB(7)), así como sustituir la instrucción ON-GOTO de la línea 210 por:

```
210 GOTO R*100+200
```

También tendremos que cambiar todos los END por STOP.

 Los bucles white-wend

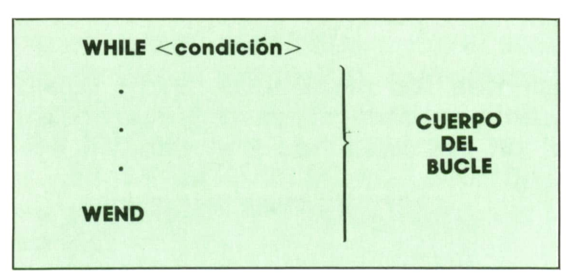
Ya hemos visto cómo podemos formar bucles condicionales combinando adecuadamente las instrucciones IF-THEN y GOTO. Sin embargo, algunos ordenadores como el AMSTRAD, el IBM o los MSX, disponen de una estructura específica para estos bucles denominada abreviadamente WHILE-WEND.

A grandes rasgos, podemos hacer las siguientes identificaciones:

WHILE - marca el comienzo del bucle.
WEND - marca el final del bucle.

Entre una instrucción WHILE y una instrucción WEND debemos situar todas las instrucciones que deseemos que se repitan y que formarán el cuerpo del bucle.

En la figura 3 podemos ver un esquema del formato general de estos bucles.



 *Esquema de bucles WHILE-WEND.*

La **condición** situada tras la instrucción WHILE puede ser de cualquiera de los tipos ya estudiados.

El funcionamiento de estos bucles es el siguiente: mientras se cumpla la condición de WHILE, se repetirán todas las instrucciones que haya hasta la instrucción WEND. Antes de que se repita el bucle, el ordenador comprueba si la condición sigue verificándose. En caso afirmativo, el bucle se ejecutará de nuevo, mientras que si la condición ya no es cierta, la ejecución pasará a la línea de programa posterior al WEND, si la hay, y si no el programa finalizará.

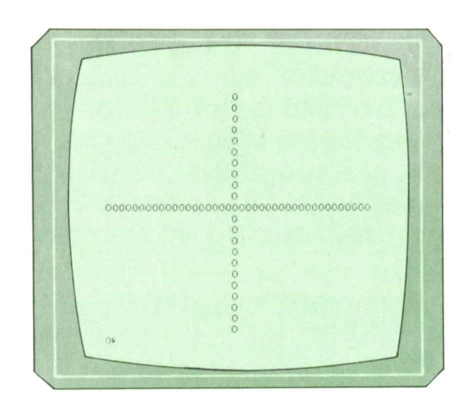
El programa 2 utiliza dos bucles WHILE-WEND para trazar una cruz en el centro de la pantalla.

```
10 REM **********
20 REM *  CRUZ  *
30 REM **********
40 CLS
50 LET C=1
60 WHILE C<=40
70 LOCATE 11,C:PRINT "O"
80 LET C=C+1
90 WEND
100 LET F=1
110 WHILE F<=22
120 LOCATE F,20:PRINT "O"
130 LET F=F+1
140 WEND
```

El primer bucle WHILE-WEND es el encargado de que se imprima en pantalla la línea horizontal, formada por oes (O) que se van imprimiendo una a una en la fila 11, desde la columna 1 hasta la 40. Esto se consigue con un contador que se incrementa en la línea 80 y que tiene como valor límite 40, gracias a la condición del WHILE (línea 60).

Análogamente el segundo bucle WHILE-WEND tiene por objeto imprimir la línea vertical en la columna 20 de la pantalla.

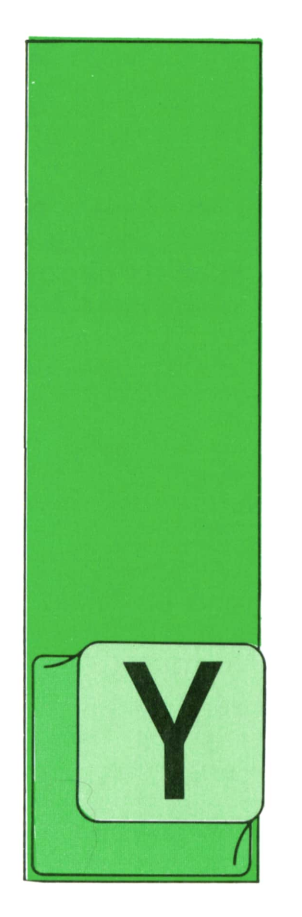
En la figura podemos ver el aspecto de la pantalla tras la ejecución.



 *Presentación en pantalla del programa 2.*

Para que este programa funcione correctamente en el AMSTRAD debemos invertir el orden de los parámetros en los LOCATE de las líneas 70 y 120.

Por otra parte, puesto que los bucles WHILE-WEND no existen en el SPECTRUM ni en el COMMODORE, no podríamos ejecutar este programa en ninguno de los dos ordenadores. De todos modos, un bucle WHILE-WEND siempre puede sustituirse por un bucle formado por IF-THEN y GOTO.



MAQUINA Z-80

SPECTRUM, AMSTRAD, MSX

Manipulaciones con datos

Ya hemos tratado de una forma general las posibles operaciones con fragmentos de 16 y de 8 bits; sin embargo, hay numerosas ocasiones en que es necesario realizar operaciones sobre un único bit, aunque éste esté contenido en una posición de memoria de 8 bits, o en un registro.

Esto se hace aún más evidente en el caso del registro de flags, en el que cada bit indica una información completamente diferente. En la tabla adjunta puede verse un resumen de cómo se ven afectados estos flags (o indicadores) por las distintas instrucciones del Z-80.

Resumen de los indicadores afectados

	D_7	Indicadores						D_0	
Instrucciones	S	Z		H		P/V	N	C	Comentarios
ADD A,s; ADC A,s	↕	↕	X	↕	X	V	0	↕	Suma o suma con arrastre de 8 bits.
SUB s; SBC A,s; CP s; NEG	↕	↕	X	↕	X	V	1	↕	Resta o resta con arrastre de 8 bits. Comparación. Cambio de signo del acumulador.
AND s	↕	↕	X	1	X	P	0	0	Operaciones lógicas.
OR s; XOR s	↕	↕	X	0	X	P	0	0	
INC s	↕	↕	X	↕	X	V	0	•	Incremento para 8 bits.
DEC s	↕	↕	X	↕	X	V	1	•	Disminución para 8 bits.
ADD DD,ss	•	•	X	X	X	•	0	↕	Suma de 16 bits.
ADC HL,ss	↕	↕	X	X	X	V	0	↕	Suma con arrastre de 16 bits.
SBC HL,ss	↕	↕	X	X	X	V	1	↕	Resta con arrastre de 16 bits.
RLA; RLCA; RRA; RRCA	•	•	X	0	X	•	0	↕	Rotación del acumulador.
RL m; RLC m; RR m; RRC m; SLA m; SRA m; SRL m	↕	↕	X	0	X	P	0	↕	Rotaciones y desplazamientos
RLD; RRD	↕	↕	X	0	X	P	0	•	Rotación de las cifras decimales a la izquierda y a la derecha
DAA	↕	↕	X	↕	X	P	•	↕	Ajuste decimal del acumulador
CPL	•	•	X	1	X	•	1	•	Complemento del acumulador
SCF	•	•	X	0	X	•	0	1	Puesta a 1 del indicador de arrastre
CCF	•	•	X	X	X	•	0	↕	Complemento del indicador de arrastre
IN r,(C)	↕	↕	X	0	X	P	0	•	Entrada de un registro
INI; IND; OUTI; OUTD	X	↕	X	X	X	X	1	•	Entrada y salida de bloques. Z a 0 si B≠0; Z a 1 en caso contrario.
INIR; INDR; OTIR; OTDR	X	1	X	X	X	X	1	•	
LDI; LDD	X	X	X	0	X	↕	0	•	Transferencia de bloques. P/V a 1 si BC≠0; P/V a 0 en caso contrario.
LDIR; LDDR	X	X	X	0	X	0	0	•	
CPI; CPIR; CPD; CPDR	X	↕	X	X	X	↕	1	•	Búsqueda en bloques. Z a 1 ai A=(HL); Z a 0 en caso contrario. P/V a 1 si BC≠0; P/V a 0 en caso contrario.
LD A,I; LD A,R	↕	↕	X	0	X	IFF	0	•	El contenido de la báscula de habilitación de interrupciones se carga en el indicador P/V.
BIT b,s	X	↕	X	1	X	X	0	•	El bit b del registro o posición s se carga en el indicador Z.

Además de la posibilidad de modificar determinados bits utilizando operaciones lógicas, el Z-80 dispone de instrucciones especializadas en manipulación de bits de forma individual. De ellas hablaremos en este apartado.

— El primer tipo de instrucciones son del tipo BIT b,a. Estas sirven para comprobar el valor de un bit de alguna palabra de 8 bits. En estas instrucciones *b* es un número del 0 al 7 que indica cuál es el bit a muestrear, y *a* indica dónde se encuentra ese bit (un registro, una posición de memoria, mediante indirección, etc.). La forma de operar de esta instrucción es la siguiente:

Pone en el flag (indicador) Z el valor del bit a considerar.

El valor en Z se suele utilizar para una condición de salta (por ejemplo, si Z=0 → JPZ,e) (de esta posibilidad se tratará en apartados posteriores).

Así, con esta instrucción podemos trasladar a Z cualquier bit, y utilizarlo luego como condición para un salto condicional.

— El segundo tipo de instrucciones es de la forma:

SET b,a

Sirven para poner a 1 el bit número *b* del byte contenido en *a*. El byte «a» puede estar indicado con los tipos de direccionamiento que pueden observarse en la siguiente tabla-resumen.

Grupo de manipulación de bits

Código mnemotécnico	Operación simbólica	Indicadores S Z H P/V N C	Códigos 76 543 210 Hex	N.º de bytes	N.º de ciclos M	N.º de estados T	Comentarios
BIT b,r	$Z \leftarrow r_b$	X ↕ X 1 X X 0 •	11 001 011 CB 01 b r	2	2	8	r Reg. 000 B 001 C 010 D 011 E 100 H 101 L 111 A
BIT b,(HL)	$Z \leftarrow \overline{(HL)}_b$	X ↕ X 1 X X 0 •	11 001 011 CB 01 b 110	2	3	12	
BIT b,(IX+d)$_b$	$Z \leftarrow \overline{(IX+d)}_b$	X ↕ X 1 X X 0 •	11 011 101 DD 11 001 011 CB ←d→ 01 b 110	4	5	20	
BIT b,(IY+d)$_b$	$Z \leftarrow \overline{(IY+d)}_b$	X ↕ X 1 X X 0 •	11 111 101 FD 11 001 011 CB ←d→ 01 b 110	4	5	20	b Bit comprob. 000 0 001 1 010 2 011 3 100 4 101 5 110 6 111 7
SET b,r	$r_b \leftarrow 1$	• • X • X • • •	11 001 011 CB [11] b r	2	2	8	
SET b,(HL)	$(HL)_b \leftarrow 1$	• • X • X • • •	11 001 011 CB [11] b 110	2	4	15	
SET b,(IX+d)	$(IX+d)_b \leftarrow 1$	• • X • X • • •	11 011 101 DD 11 00! 011 CB ←d→ [11] b 110	4	6	23	
SET b,(IY+d)	$(IY+d)_b \leftarrow 1$	• • X • X • • •	11 111 101 FD !1 001 012 CB ←d→ [11] b 110	4	6	23	
RES b,m	$m_b \rightarrow 0$ m≡r,(HL),(IX+d),(IY+d)	• • X • X • • •	[10]				El código se forma como en las SET b,m, pero reemplazando [11] por [10]; indicadores y estados como para SET

NOTA: m_b representa el bit b (0 a 7) del registro o la posición m.

Evidentemente, *b* es un número del 0 al 7, para indicar cuál es el bit que poner a 1.

— El último tipo de instrucciones es de la forma: Res b,a.

Esta instrucción realiza la operación opuesta, es decir, pone a cero el bit *b* del byte indicado por *a*.

Las funciones de *b* y *a* son similares a las de la instrucción anterior.



Programas de ejemplo

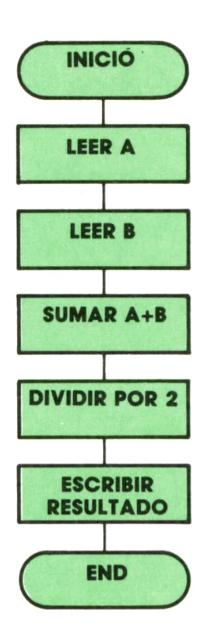
Hasta aquí hemos explicado ya las suficientes instrucciones del ensamblador del Z-80 como para atrevernos a hacer una primera incursión en la programación en lenguaje máquina.

Vamos a realizar un primer programa que realice la media aritmética de dos números de 8 bits.

1.º Fijemos el algoritmo a utilizar:

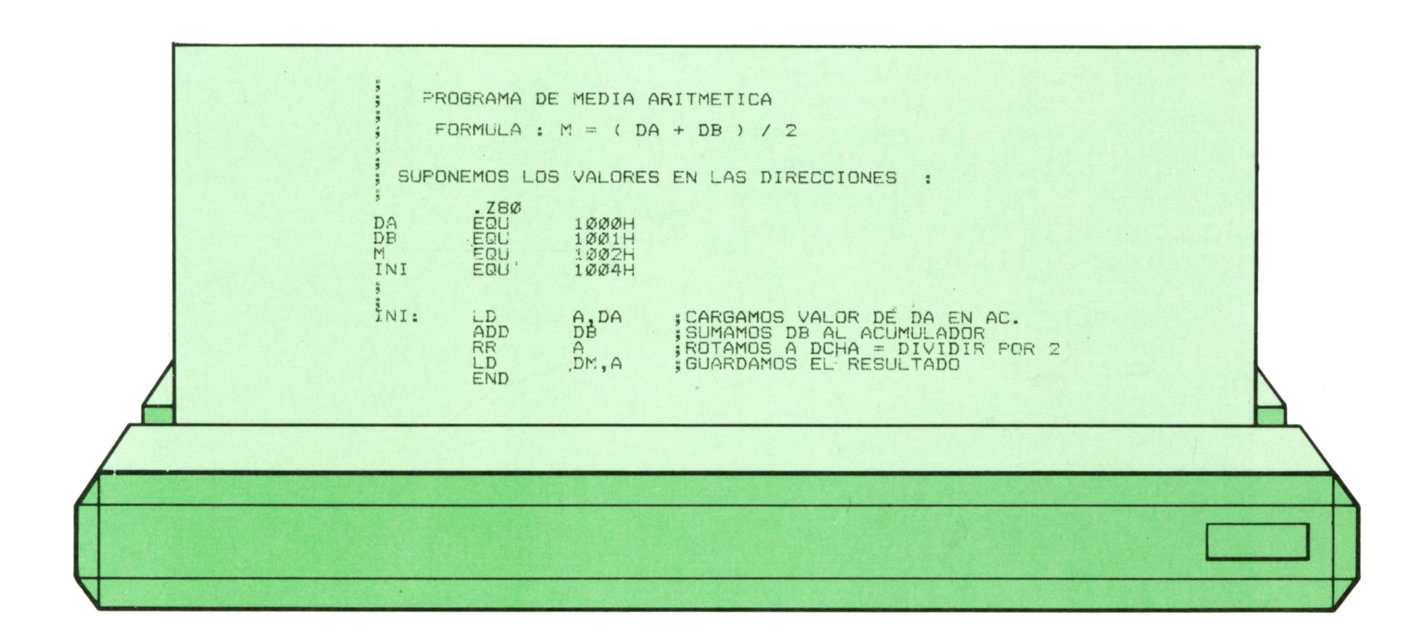
$$\text{Resultado} = \frac{\text{Operando A} + \text{Operando D}}{2}$$

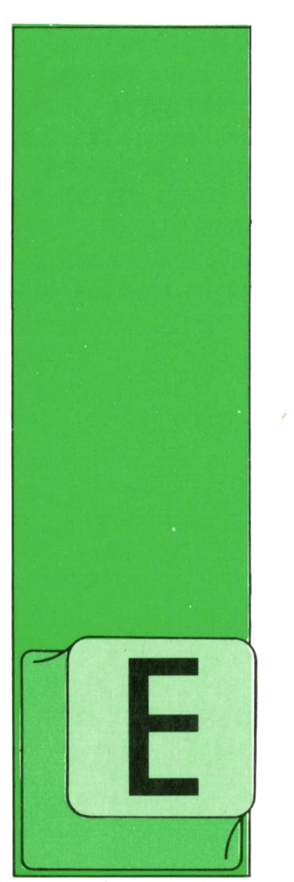
2.º A continuación realicemos un pequeño organigrama para fijar ideas:



En la programación en lenguaje máquina debemos desmenuzar las operaciones hasta que seamos capaces de intuir claramente cómo sustituir cada paso por una o varias instrucciones. Dependiendo de la capacidad y sobre todo de la experiencia del programador, estos bloques serán más o menos detallados.

3.° Realicemos el programa en ensamblador.





PROGRAMAS

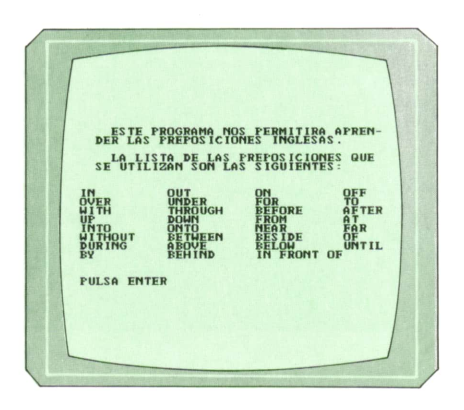
EDUCATIVOS • DE UTILIDAD • DE GESTION • DE JUEGOS



Programa: Preposiciones inglesas

EL programa de PREPOSICIONES INGLESAS que vamos a ver a continuación nos va a permitir repasar nuestros conocimientos sobre las mismas.

Las preposiciones que el programa nos permite repasar son las que aparecen en la figura 1.



```
   ESTE PROGRAMA NOS PERMITIRA APREN-
  DER LAS PREPOSICIONES INGLESAS.

   LA LISTA DE LAS PREPOSICIONES QUE
  SE UTILIZAN SON LAS SIGUIENTES:

 IN          OUT        ON           OFF
 OVER        UNDER      FOR          TO
 WITH        THROUGH    BEFORE       AFTER
 UP          DOWN       FROM         AT
 INTO        ONTO       NEAR         FAR
 WITHOUT     BETWEEN    BESIDE       OF
 DURING      ABOVE      BELOW        UNTIL
 BY          BEHIND     IN FRONT OF

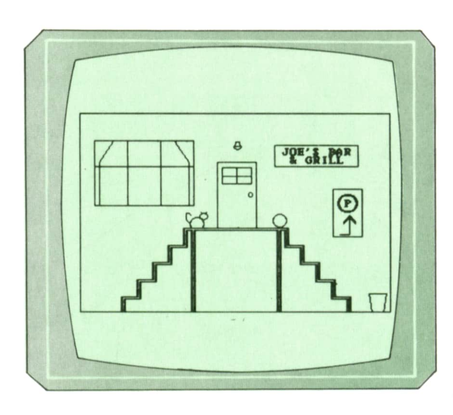
 PULSA ENTER
```



Lista de preposiciones que utiliza el programa.

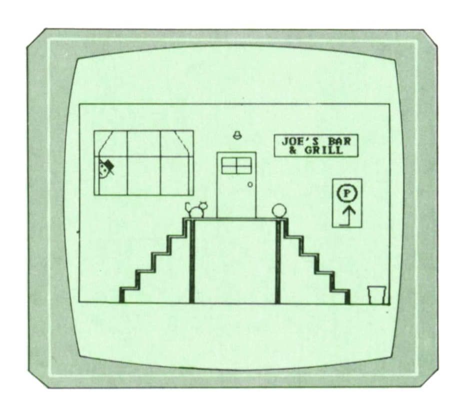
La forma del programa no es la que suelen tener los programas educativos. En este caso, como el usuario suele ser un niño, se ha intentado que la parte gráfica sea muy llamativa para no aburrir al usuario.

Lo único que hace falta para ver la calidad del programa es ver las pantallas que aparecen del mismo en plena ejecución.



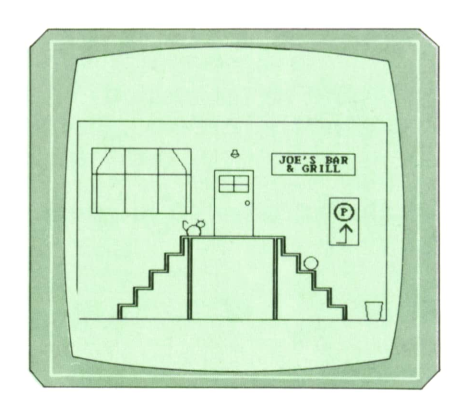


Primera escena.





Segunda escena. Joe se asoma por la ventana.





Tercera escena. La pelota cae por las escaleras.

No es necesario dar ninguna instrucción de uso, ya que las pocas que se necesitan aparecen por pantalla.

```
1000 REM **********************************
1010 REM *                                *
1020 REM *     PREPOSICIONES INGLESAS     *
1030 REM *                                *
1040 REM *  Por: Peter Bergmann           *
1050 REM *                                *
1060 REM **********************************
1070 REM *                                *
1080 REM *           VARIABLES            *
1090 REM *           ---------            *
1100 REM *  U$   -  ARRAY DE RESPUESTAS   *
1110 REM *  L$   -  ARRAY DE RESULTADOS   *
1120 REM *  B$   -  COMIENZO DE FASE      *
1130 REM *  E$   -  FIN DE FASE           *
1140 REM *  P$   -  PREPOSICION CORRECTA  *
1150 REM * T$,O$ - PREPOSICIONES FALSAS   *
1160 REM *  T    -  NUMERO DE PANTALLA    *
1170 REM *  N    -  NUM. DE FASES EN PANT. *
1180 REM *  ND   -  NUM. DE FRASES. FIN P. *
1190 REM *  A$   -  RESPUESTA (1-3)       *
1200 REM *  A    -  RESPUESTA (1-3)       *
1210 REM *  R$   -  RESPUESTA (S/N)       *
1220 REM *  C$   -  RESPUESTA (CONTINUAR) *
1230 REM *  X$   -  FLAG PARA CONTINUAR   *
1240 REM *  RN   -  NUMERO ALEATORIO      *
1250 REM *  W    -  RESPUESTA CORRECTA    *
1260 REM *  HZ   -  FRECUENCIA DE SONIDO  *
1270 REM * I,J   -  CONTADORES            *
1280 REM *  NC   -  NUMERO CORRECTO       *
1290 REM *  NF   -  NUMERO FALSO          *
1300 REM *  PC   -  PERCENTAJE CORRECTO   *
1310 REM *  PF   -  PERCENTAJE FALSO      *
1320 REM *                                *
1330 REM **********************************
1340 REM
1350 REM *** CABECERA ***
1360 REM
1370 CLS
1380 KEY OFF
1390 PRINT STRING$(40,"*")
1400 FOR I = 1 TO 19
1410    PRINT  "*"; TAB(40); "*"
1420 NEXT I
1430 PRINT STRING$(40,"*")
1440 LOCATE 8,10: PRINT "ENGLISH PREPOSITIONS"
1450 LOCATE 9,9: PRINT "----------------------"
1460 LOCATE 12,7: PRINT "(c) Ed. Siglo Cultural, 1987"
1470 FOR I = 1 TO 1000
1480 NEXT I
1490 REM
1500 REM *** DECLARACION DE ARRAYS ***
1510 REM
1520 DIM U$(35)
1530 DIM L(4,2)
1540 REM
1550 REM *********************
1560 REM * PROGRAMA PRINCIPAL *
1570 REM *********************
1580 REM
1590 CLS
1600 LET X$ = "C"
1610 GOSUB 1710
```

```
1620 GOSUB 2080
1630 IF X$ = "C" THEN GOSUB 2300
1640 IF X$ = "C" THEN GOSUB 2570
1650 IF X$ = "C" THEN GOSUB 2770
1660 GOSUB 4080
1670 CLS
1680 PRINT "ADIOS..."
1690 KEY ON
1700 END
1710 REM
1720 REM ****************
1730 REM * INTRODUCCION *
1740 REM ****************
1750 REM
1760 CLS
1770 PRINT "    ESTE PROGRAMA NOS PERMITIRA APREN-"
1780 PRINT "  DER LAS PREPOSICIONES INGLESAS."
1790 PRINT
1800 PRINT "    LA LISTA DE LAS PREPOSICIONES QUE"
1810 PRINT "  SE UTILIZAN SON LAS SIGUIENTES:"
1820 PRINT :PRINT
1830 PRINT "IN          OUT         ON          OFF   "
1840 PRINT "OVER        UNDER       FOR         TO    "
1850 PRINT "WITH        THROUGH     BEFORE      AFTER"
1860 PRINT "UP          DOWN        FROM        AT"
1870 PRINT "INTO        ONTO        NEAR        FAR   "
1880 PRINT "WITHOUT     BETWEEN     BESIDE      OF   "
1890 PRINT "DURING      ABOVE       BELOW       UNTIL"
1900 PRINT "BY          BEHIND      IN FRONT OF      "
1910 PRINT :PRINT
1920 PRINT "PULSA ENTER"
1930 C$ = INPUT$(1)
1940 RETURN
1950 REM
1960 REM *** DECLARACION DE ARRAYS ***
1970 REM
1980 FOR I = 1 TO 35
1990    LET U$(I) = "0"
2000 NEXT I
2010 IF T <> 1 THEN GOTO 2070
2020 FOR I = 1 TO 4
2030    FOR J = 1 TO 2
2040       LET L(I,J) = 0
2050    NEXT J
2060 NEXT I
2070 RETURN
2080 REM
2090 REM ************************************
2100 REM * CONOCIENDO EL JOE'S BAR AND GRILL *
2110 REM ************************************
2120 REM
2130 CLS
2140 WIDTH 40
2150 PRINT "   PANTALLA 1 - JOE'S BAR AND GRILL    "
2160 PRINT "  ----------------------------------   "
2170 PRINT
2180 PRINT "   BENVENIDO A 'JOE'S BAR AND GRILL'! "
2190 PRINT " ESTE BAR ESTA EN NUEVA YORK CERCA DE "
2200 PRINT " MANHATTAN. MIRA A 'JOE'S' Y TERMINA  "
2210 PRINT " LAS FRASES.                          "
2220 PRINT :PRINT
2230 PRINT "PULSA ENTER"
2240 C$ = INPUT$(1)
2250 CLS
2260 LET T = 1
2270 LET N = 31
2280 GOSUB 2980
2290 RETURN
```

```
2300 REM
2310 REM *******************
2320 REM * CONOCIENDO A JOE *
2330 REM *******************
2340 REM
2350 CLS
2360 WIDTH 40
2370 PRINT "         PANTALLA 2 - JOE            "
2380 PRINT "        ------------------           "
2390 PRINT
2400 PRINT "   AHORA, TE ENCUENTRAS A JOE. JOE ES "
2410 PRINT " MUY TIMIDO. EL NUNCA SALE DE CASA. EL"
2420 PRINT " VIVE SOLO, Y USUALMENTE SOLO HABLA   "
2430 PRINT " CON SU GATO. PERO JOE SIEMPRE TIENE  "
2440 PRINT " COMIDA PARA LA GENTE. EL BAR ESTA ABI"
2450 PRINT " ERTO DESDE LA 1 PM HASTA LA 1 AM. HAY"
2460 PRINT " MUCHA GENTE EN JOE'S POR LA TARDE A  "
2470 PRINT " LA HORA DE COMER.                    "
2480 PRINT "   MIRA A JOE Y TERMINA LAS FRASES.   "
2490 PRINT :PRINT
2500 PRINT "PULSA ENTER"
2510 C$ = INPUT$(1)
2520 CLS
2530 LET T = 2
2540 LET N = 15
2550 GOSUB 2980
2560 RETURN
2570 REM
2580 REM *****************************
2590 REM * CONOCIENDO EL BALON DE JOE *
2600 REM *****************************
2610 REM
2620 CLS
2630 WIDTH 40
2640 PRINT "   PANTALLA 3 - EL BALON DE JOE      "
2650 PRINT "  ------------------------------     "
2660 PRINT
2670 PRINT "   JOE TIENE UN BALON. FIJATE EN EL  '
2680 PRINT " Y TERMINA LAS FRASES.               "
2690 PRINT :PRINT
2700 PRINT "PULSA ENTER"
2710 C$ = INPUT$(1)
2720 CLS
2730 LET T = 3
2740 LET N = 16
2750 GOSUB 2980
2760 RETURN
2770 RE
2780 REM *****************
2790 REM * EL GATO DE JOE *
2800 REM *****************
2810 REM
2820 CLS
2830 WIDTH 40
2840 PRINT "    PANTALLA 4 - EL GATO DE JOE      "
2850 PRINT "   ---------------------------        "
2860 PRINT
2870 PRINT "    JOE TAMBIEN TIENE UN GATO. FIJATE "
2880 PRINT " LO UQE HACE Y TERMINA LAS FRASES.    "
2890 PRINT :PRINT
2900 PRINT "PULSA ENTER"
2910 C$ = INPUT$(1)
2920 CLS
2930 LET T = 4
2940 LET N = 13
2950 GOSUB 2980
2960 RETURN
2970 REM
```

```
2980 REM ********************************
2990 REM * RUTINA DE CREACION DE FRASES *
3000 REM ********************************
3010 REM
3020 RANDOMIZE TIMER
3030 GOSUB 1950
3040 GOSUB 4340
3050 LET ND = 0
3060 IF ND < N THEN GOSUB 3230 ELSE GOTO 3160
3070 ND = ND + 1
3080 GOSUB 3380
3090 GOSUB 3670
3100 LOCATE 23,10: PRINT "(QUIERES OTRA (S/N)";
3110 INPUT R$
3120 IF R$ = "S" THEN GOTO 3060
3130 IF R$ <> "N" THEN GOTO 3100
3140 GOSUB 3960
3150 IF T = 4 THEN GOTO 3210
3160 LOCATE 23,10: PRINT "(QUIERES VER OTRA SITUACION (S/N)";
3170 INPUT R$
3180 IF R$ = "S" THEN RETURN
3190 IF R$ <> "N" THEN GOTO 3160
3200 LET X$ = "S"
3210 RETURN
3220 REM
3230 REM *****************
3240 REM * BUSCAR FRASES *
3250 REM *****************
3260 REM
3270 IF T = 1 THEN RESTORE 6360
3280 IF T = 2 THEN RESTORE 6700
3290 IF T = 3 THEN RESTORE 6880
3300 IF T = 4 THEN RESTORE 7050
3310 RN = INT(RND*N) + 1
3320 IF U$(RN) <> "O" THEN GOTO 3310
3330 FOR I = 1 TO RN
3340    READ B$,E$,P$,T$,O$
3350 NEXT I
3360 RETURN
3370 REM
3380 REM ***********************************
3390 REM * IMPRIMIR FRASE Y LEER RESPUESTA *
3400 REM ***********************************
3410 REM
3420 ON KEY(1) GOSUB 6140
3430 KEY(1) OFF
3440 SCREEN 0
3450 CLS
3460 WIDTH 80
3470 W = INT(RND*3) + 1
3480 LOCATE 11,10: PRINT B$;"______";E$
3490 LOCATE 13,35: PRINT "1. ";
3500 IF W = 1 THEN PRINT P$ ELSE PRINT T$
3510 LOCATE 14,35: PRINT "2. ";
3520 IF W = 2 THEN PRINT P$: GOTO 3540
3530 IF W = 1 THEN PRINT T$ ELSE PRINT O$
3540 LOCATE 15,35: PRINT "3. ";
3550 IF W = 3 THEN PRINT P$ ELSE PRINT O$
3560 LOCATE 20,25: PRINT "(TU RESPUESTA (1-3)?";
3570 LOCATE 23,2: PRINT "PULSA 'F1' PARA VER PANTALLA"
3580 KEY(1) ON
3590 A$ = INKEY$: IF A$ = "" THEN GOTO 3590
3600 KEY(1) OFF
3610 IF LEN(A$) > 1 THEN GOTO 3580
3620 IF (ASC(A$) < 49) OR (ASC(A$) > 51) THEN GOTO 3580
3630 A = VAL(A$)
3640 LET U$(RN) = "X"
3650 RETURN
```

```
3660 REM
3670 REM **********************
3680 REM * CHEQUEAR RESPUESTA *
3690 REM **********************
3700 REM
3710 CLS
3720 IF W = A THEN GOTO 3750
3730 LOCATE 4,36: PRINT "FALSO!"
3740 SOUND 150, 15
3750 LOCATE 8,10: PRINT "LA RESPUESTA CORRECTA ES:";
3760 LOCATE 13,35: PRINT "1. ";
3770 IF W = 1 THEN PRINT P$ ELSE PRINT T$
3780 LOCATE 14,35: PRINT "2. ";
3790 IF W = 2 THEN PRINT P$: GOTO 3810
3800 IF W = 1 THEN PRINT T$ ELSE PRINT O$
3810 LOCATE 15,35: PRINT "3. ";
3820 IF W = 3 THEN PRINT P$ ELSE PRINT O$
3830 LOCATE 18,10: PRINT B$;P$;E$
3840 FOR I = 1 TO 4
3850    IF W = A THEN LOCATE 4,30: PRINT "
3860    LOCATE 12+W,38: PRINT "                  "
3870    IF W = A THEN SOUND 1000, 6
3880    FOR J = 1 TO 400: NEXT J
3890    IF W = A THEN LOCATE 4,30: PRINT "-CORRECTO!"
3900    LOCATE 12+W,38: PRINT P$
3910    FOR J = 1 TO 400: NEXT J
3920 NEXT I
3930 IF W = A THEN LET U$(RN) = "C"
3940 RETURN
3950 REM
3960 REM ******************************
3970 REM * TOTAL ACIERTOS POR SECCION *
3980 REM ******************************
3990 REM
4000 LET NC = 0
4010 FOR I = 1 TO N
4020    IF U$(I) = "C" THEN NC = NC + 1
4030 NEXT I
4040 LET L(T,1) = ND
4050 LET L(T,2) = NC
4060 RETURN
4070 REM
4080 REM *******************************
4090 REM * IMPRIMIR RESULTADOS FINALES *
4100 REM *******************************
4110 REM
4120 CLS
4130 WIDTH 40
4140 LOCATE 2,12: PRINT "RESULTADOS FINALES"
4150 LOCATE 3,11: PRINT "--------------------"
4160 LOCATE 5,1: PRINT "SEC.  NUM.  NUM.  NUM.    %       %      "
4170 LOCATE 6,1: PRINT "NUM.  TOT.  COR.  FAL.    COR.    FAL.   "
4180 LOCATE 7,1: PRINT "----  ----  ----  ----  ------- ------- "
4190 FOR I = 1 TO 4
4200    LET ND = L(I,1)
4210    LET NC = L(I,2)
4220    NF = ND - NC
4230    IF ND = 0 THEN LET PC = 0: LET PF = 0: GOTO 4260
4240    PC = (NC/ND)*100
4250    PF = (NF/ND)*100
4260    LOCATE 7+I,2: PRINT USING "##    ";I;ND;NC;NF
4270    LOCATE 7+I,25: PRINT USING "###.##  ";PC;PF
4280    LOCATE 7+I,31: PRINT "%": LOCATE 7+I,39: PRINT "%"
4290 NEXT I
4300 LOCATE 20,2: PRINT "PULSA ENTER"
4310 C$ = INPUT$(1)
4320 RETURN
4330 REM
```

```
4340 REM ***********
4350 REM * DIBUJOS *
4360 REM ***********
4370 REM
4380 CLS
4390 SCREEN 1
4400 GOSUB 4570
4410 GOSUB 4850
4420 GOSUB 4940
4430 GOSUB 5020
4440 GOSUB 5380
4450 GOSUB 5530
4460 IF T = 3 THEN GOSUB 5090
4470 IF T = 3 THEN GOSUB 5590
4480 IF T = 4 THEN GOSUB 5770
4490 IF T = 2 THEN GOSUB 5990
4500 C$ = INPUT$(1)
4510 SCREEN 0
4520 WIDTH 80
4530 RETURN
4540 REM
4550 REM *** DIBUJAR ESCALERAS ***
4560 REM
4570 LOCATE 14,15: PRINT CHR$(203);
4580 FOR I = 1 TO 10
4590 PRINT CHR$(205);
4600 NEXT I
4610 PRINT CHR$(203)
4620 FOR J = 15 TO 26 STEP 11
4630    FOR I = 15 TO 23
4640       LOCATE I,J: PRINT CHR$(186)
4650    NEXT I
4660 NEXT J
4670 FOR I = 0 TO 8 STEP 2
4680    LOCATE 14+I,14-I: PRINT CHR$(201)
4690    LOCATE 14+I,27+I: PRINT CHR$(187)
4700    LOCATE 15+I,14-I: PRINT CHR$(186)
4710    LOCATE 15+I,27+I: PRINT CHR$(186)
4720 NEXT I
4730 FOR I = 2 TO 8 STEP 2
4740    LOCATE 14+I,15-I: PRINT CHR$(205);CHR$(188)
4750    LOCATE 14+I,25+I: PRINT CHR$(200);CHR$(205)
4760 NEXT I
4770 CIRCLE (205,99),7
4780 CIRCLE (120,102),7,3,5.6,3.7
4790 CIRCLE (127,95),4
4800 DRAW "BM125,92"
4810 DRAW "S2 H2 BM129,92 E2"
4820 DRAW "BM112,101"
4830 DRAW "S4 H3 U6 R3 F2"
4840 RETURN
4850 REM
4860 REM *** DIBUJAR PUERTA ***
4870 REM
4880 DRAW "BM141,105"
4890 DRAW "U60 R40 D60"
4900 DRAW "BU40 BL5"
4910 DRAW "L30 U15 R30 D15 L15 U15 R15 D8 L30"
4920 CIRCLE (175,75),2
4930 RETURN
4940 REM
4950 REM *** DIBUJAR EDIFICIO ***
4960 REM
4970 DRAW "BM0,0"
4980 DRAW "D184 R319 U184 L319"
4990 DRAW "BM306,184"
5000 DRAW "L6 M297,167 R18 M312,184"
5010 RETURN
```

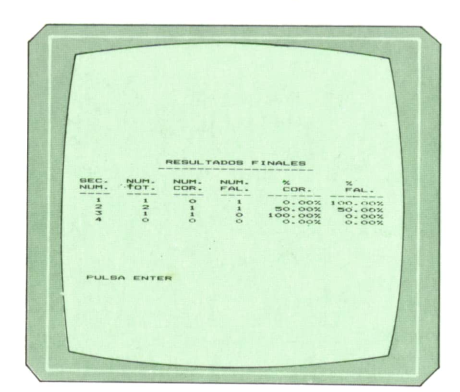
```
5020 REM
5030 REM *** DIBUJAR VENTANA ***
5040 REM
5050 DRAW "BM15,25"
5060 DRAW "D60 R102 U60 L102 D24 R102 U24 L34 D60 L34 U60 L14"
5070 DRAW "M20,45 D40 BM15,25 R82 M112,45 D40"
5080 RETURN
5090 REM
5100 REM *** DIBUJAR BALON ***
5110 REM
5120 LET HZ = 1000
5130 FOR I = 0 TO 4
5140    CIRCLE (205+I*18,99+I*16),7
5150    FOR J = 1 TO 400: NEXT J
5160    CIRCLE (205+I*18,99+I*16),7,0
5170 SOUND HZ,5
5180 HZ = HZ - 150
5190 NEXT I
5200 CIRCLE (288,177),7
5210 FOR J = 1 TO 100: NEXT J
5220 CIRCLE (288,177),7,0
5230 PLAY "T150 MB ML O3 L16 CDEFGAB"
5240 DRAW "BM288,177 C3 U40"
5250 CIRCLE (297,137),9,3,0,3.14
5260 DRAW "BM288,177 C0 U40"
5270 CIRCLE (297,137),9,0,0,3.14
5280 CIRCLE (306,144),7
5290 PLAY "T90 MB ML O3 L16 BAGFEDC"
5300 FOR I = 1 TO 16
5310    CIRCLE (306,143+I),7,0
5320    CIRCLE (306,144+I),7
5330 NEXT I
5340 CIRCLE (306,160),7,0
5350 SOUND 75,7
5360 FOR I = 1 TO 800: NEXT I
5370 RETURN
5380 REM
5390 REM *** DIBUJAR CARTEL ***
5400 REM
5410 DRAW "BM200,29"
5420 DRAW "D20 R86 U20 L86"
5430 LOCATE 5,27: PRINT "JOE'S BAR"
5440 LOCATE 6,27: PRINT " & GRILL "
5450 DRAW "BM260,70"
5460 DRAW "D44 R30 U44 L30"
5470 LOCATE 11,35: PRINT "P"
5480 CIRCLE (275,83),10
5490 DRAW "BM267,112"
5500 DRAW "R10 U16 F6 U2 H6 G6 D2 E6"
5510 CIRCLE (275,83),9
5520 RETURN
5530 REM
5540 REM *** DIBUJAR LAMPARA ***
5550 REM
5560 CIRCLE (162,30),4,3,3.14,6.28
5570 DRAW "L4 E2 U2 R4 D2 F2 L6"
5580 RETURN
5590 REM*
5600 REM* BOUNCE BALL UP
5610 REM*
5620 SOUND 1000,1
5630 PLAY "T160 MB ML O3 L16 CDEFEDC"
5640 CIRCLE (291,166),14,3,0,3.14
5650 FOR I = 1 TO 90: NEXT I
5660 CIRCLE (291,166),14,0,0,3.14
5670 LET HZ = 250
5680 FOR I = 4 TO 0 STEP -1
5690   SOUND HZ,5
```

```
5700   CIRCLE (205+I*18,99+I*16),7
5710   FOR J = 1 TO 400: NEXT J
5720   IF I = 0 THEN GOTO 5750
5730   CIRCLE (205+I*18,99+I*16),7,0
5740    HZ = HZ + 150
5750 NEXT I
5760 RETURN
5770 REM*
5780 REM* DRAW KITTY
5790 REM*
5800 FOR I = 1 TO 2000: NEXT I
5810 CIRCLE (120,102),7,0,5.6,3.7
5820 CIRCLE (127,95),4,0
5830 DRAW "BM125,92"
5840 DRAW "C0 S2 H2 BM129,92 E2"
5850 DRAW "BM112,101"
5860 DRAW "C0 S4 H3 U6 R3 F2"
5870 PLAY "T120 MB ML L16 O4 BAGFEDC O3 BAGFEDC O2 BAGFEDC O1 BAGFDEC"
5880 FOR I = 115 TO 177 STEP 2
5890    CIRCLE (105+INT(I/5),I),7,1
5900    CIRCLE (105+INT(I/5),I),7,0
5910 NEXT I
5920 SOUND 37,15
5930 DRAW "BM135,178"
5940 DRAW "C3 U4 L2 BM145,178 U4 L2"
5950 CIRCLE (140,183),7,3,0,3.14
5960 CIRCLE (151,183),4,3,0,3.14
5970 DRAW "BM133,183 H4 U7"
5980 RETURN
5990 REM
6000 REM *** DIBUJAR A JOE ***
6010 REM
6020 CIRCLE (21,62),9,3,4.71,1.57
6030 DRAW "BE5 H5 F14 H4 E4 H6 G4 E1 F6 E1 H6 E1 F6"
6040 CIRCLE (23,58),1
6050 CIRCLE (26,62),1
6060 DRAW "BM21,64 F2"
6070 FOR I = 1 TO 1000: NEXT I
6080 CIRCLE (21,62),9,0,4.71,1.57
6090 DRAW "C0 BE5 H5 F14 H4 E4 H6 G4 E1 F6 E1 H6 E1 F6"
6100 CIRCLE (23,58),1,0
6110 CIRCLE (26,62),1,0
6120 DRAW "C0 BM21,64 F2 C3"
6130 RETURN
6140 REM
6150 REM *** INTERRUPCIONES CON F1 ***
6160 REM
6170 GOSUB 4340
6180 SCREEN 0
6190 CLS
6200 WIDTH 80
6210 LOCATE 11,10: PRINT B$;"_____";E$
6220 LOCATE 13,35: PRINT "1. ";
6230 IF W = 1 THEN PRINT P$ ELSE PRINT T$
6240 LOCATE 14,35: PRINT "2. ";
6250 IF W = 2 THEN PRINT P$: GOTO 3540
6260 IF W = 1 THEN PRINT T$ ELSE PRINT O$
6270 LOCATE 15,35: PRINT "3. ";
6280 IF W = 3 THEN PRINT P$ ELSE PRINT O$
6290 LOCATE 20,25: PRINT "(SU RESPONCIA (1-3)?";
6300 LOCATE 23,3: PRINT "PULSO 'F1' VER LA VISTA"
6310 KEY(1) ON
6320 RETURN
6330 REM
6340 REM *** DATA PARA EL EDIFICIO ***
6350 REM
6360 DATA "There are curtains "," the window.","in","on","at"
6370 DATA "The sign 'Joe's Bar & Grill' is "," the building.","on","off","at"
```

```
6380 DATA "A light is "," the door.","over","under","below"
6390 DATA "The door is "," the light.","under","over","above"
6400 DATA "The stairs go "," the door.","to","at","of"
6410 DATA "The stairs go "," the street to the door.","from","to","at"
6420 DATA "The light is turned on "," the night.","during","at","of"
6430 DATA "The parking sign is "," the stairs.","beside","with","from"
6440 DATA "The 'G' in 'GRILL' comes "," the 'R'.","before","after","during"
6450 DATA "The 'R' in 'GRILL' comes "," the 'G'.","after","before","during"
6460 DATA "The door is "," the sign and the window.","between","above","during"
6470 DATA "The stairs are "," the building.","in front of","behind","before"
6480 DATA "The door is "," the cat.","near","far from","after"
6490 DATA "The ball is "," the top step.","on","off","at"
6500 DATA "The door is "," the light.","below","above","over"
6510 DATA "The light is "," the door.","above","below","under"
6520 DATA "The ball is "," the door.","near","far from","between"
6530 DATA "The parking sign points to parking "," the building.","behind","in fr
ont of","between"
6540 DATA "The stairs go "," the door to the street.","from","to","between"
6550 DATA "The cat is "," the building.","in front of","behind","between"
6560 DATA "The 'I' in 'GRILL' is "," the 'R' and the 'L'.","between","before","a
fter"
6570 DATA "The 'R' in 'BAR' is "," the 'A'.","after","after","between"
6580 DATA "The 'A' in 'BAR' is "," the 'R'.","before","after","during"
6590 DATA "The parking sign is "," the 'JOE'S' sign.","below","above","over"
6600 DATA "The 'JOE'S' sign is "," the parking sign.","above","below","under"
6610 DATA "The door is "," the window.","beside","far from","between"
6620 DATA "The door is "," the window.","near","far from","into"
6630 DATA "There is the letter 'L' "," the word 'GRILL'.","in","on","over"
6640 DATA "The door is "," the street.","far from","below","beside"
6650 DATA "The parking sign is "," the 'JOE'S' sign.","under","over","above"
6660 DATA "The 'JOE'S' sign is "," the parking sign.","over","below","under"
6670 REM
6680 REM *** DATA PARA JOE ***
6690 REM
6700 DATA "Joe was looking "," the window.","through","at","in"
6710 DATA "Joe was looking "," you.","at","through","in"
6720 DATA "Joe was hiding "," the curtain.","behind","in front of","under"
6730 DATA "Joe is shy. He is afraid "," most people.","of","at","by"
6740 DATA "Joe never goes "," the door.","out","in","for"
6750 DATA "Joe always has good food "," the customers.","for","by","to"
6760 DATA "Joe has a hat "," his head.","on","off","by"
6770 DATA "Joe often talks "," his cat.","to","at","for"
6780 DATA "Joe is never "," his hat.","without","with","for"
6790 DATA "Joe's bar is open "," 1am.","until","by","to"
6800 DATA "Joe doesn,t open the bar "," 1pm.","until","at","to"
6810 DATA "There are many customers "," dinnertime.","during","at","to"
6820 DATA "Joe sometimes talks "," the customers.","with","without","by"
6830 DATA "Joe cooks the food "," the customers.","for","at","by"
6840 DATA "Joe lives "," himself.","by","at","near"
6850 REM
6860 REM *** DATA PARA LA PELOTA ***
6870 REM
6880 DATA "First the ball bounced "," the stairs.","down","up","through"
6890 DATA "After bouncing down the stairs, the ball bounced "," the street.","on
to","into","near"
6900 DATA ""," bouncing down the stairs, the ball bounced onto the street.","Aft
er","Before","During"
6910 DATA "Before bouncing up the stairs, the ball bounced "," the bucket.","int
o","onto","far from"
6920 DATA ""," bouncing up the stairs, the ball bounced into the bucket.","Befor
e","After","During"
6930 DATA "The ball started at the top "," the stairs.","of","near","by"
6940 DATA "The ball started "," the top of the stairs.","at","to","by"
6950 DATA "After the ball bounced into the bucket, the ball bounced "," the stai
rs.","up","down","through"
6960 DATA ""," the ball bounced into the bucket, the ball bounced up the stairs.
","After","Before","During"
6970 DATA "After the ball bounced "," the bucket, the ball bounced up the stairs
```

```
.","into","onto","at"
6980 DATA "The ball bounced on each one "," the stairs.","of","onto","by"
6990 DATA "Before bouncing into the bucket, the ball bounced "," the street.","o
nto","into","by"
7000 DATA ""," bouncing into the bucket, the ball bounced onto the street.","Bef
ore",After","During"
7010 DATA "Before bouncing "," the bucket, the ball bounced onto the street.","i
nto","onto","by"
7020 DATA "Before bouncing "," the stairs, the ball bounced into the bucket.","u
p","down","by"
7030 DATA "After bouncing "," the stairs, the ball bounced onto the street.","do
wn","up","near"
7040 REM
7050 REM *** DATA PARA EL GATO ***
7060 REM
7070 DATA "The cat was sitting "," the top stair.","on","at","by"
7080 DATA "The cat was sitting "," the top of the stairs.","at","in","of"
7090 DATA "The cat was sitting on the top "," the stairs.","of","to","at"
7100 DATA "The cat fell "," the stairs.","off","on","of"
7110 DATA "The cat landed "," the stairs.","under","above","over"
7120 DATA "The fall to the ground was very bad "," the cat.","for","to","from"
7130 DATA "The fall "," the ground was vary bad for the cat.","to","at","for"
7140 DATA "The cat landed "," a loud noise.","with","without","for"
7150 DATA "The cat fell "," the top step.","from","at","for"
7160 DATA "The cat fell "," the street.","onto","into","at"
7170 DATA "After the fall, the cats legs are pointing "," the sky.","at","for","
on"
7180 DATA ""," the fall, the cats legs are pointing at the sky.","After","Before
","During"
7190 DATA "The cat landed "," the post.","beside","far from","over"
```

Al final del programa aparece una tabla que nos informa de las respuestas correctas y erróneas que hemos tenido en cada una de las cuatro fases de las que se compone el programa.




Resultados finales.

Este programa aparecerá aproximadamente en versiones para los demás ordenadores.

Programa: Carrera de caballos para Spectrum

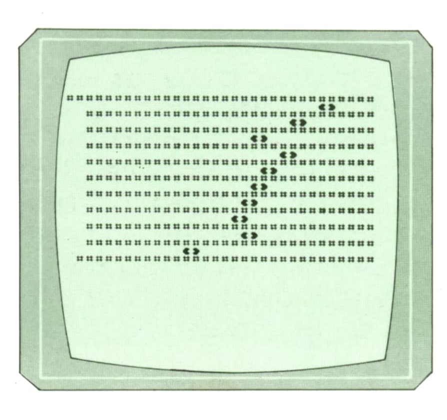
El juego que aparece a continuación nos va a permitir trasladarnos al hipódromo de la Zarzuela sin tener que salir de casa. Con este programa podremos ver una serie de carreras de caballos e incluso podremos apostar por el que más nos guste.



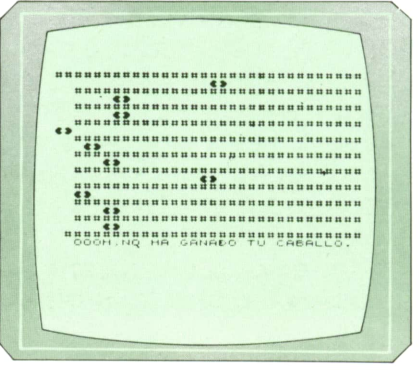

Presentación del programa.

```
 10 REM ************************
 20 REM * CARRERA DE CABALLOS *
 30 REM ************************
 31 REM *   POR CARLOS DORAL   *
 32 REM ************************
 34 REM
 40 GO SUB 970
 50 CLS
 60 DIM c(10)
 70 PRINT AT 0,6; FLASH 1;"CARRERA DE CABALLOS"
 80 PRINT '
 90 PRINT "  Dispones de 2000 pts. para ju- gar,juega y gana apostando al   cab
allo que creas conveniente."
100 PRINT
110 PRINT "  Segun lo que apuestes,asi sera el bote,y eso ganaras."
120 PRINT ''' FLASH 1;"    PULSA UNA TECLA PARA JUGAR    "
130 FOR f=1 TO 50
140 NEXT f
150 IF INKEY$="" THEN  GO TO 150
160 LET pt=2000
170 CLS
180 PRINT AT 10,10;"Tienes= ";pt
190 INPUT "Cuanto quieres apostar ? ";ap
200 IF ap<1 OR ap>pt THEN  GO TO 190
210 RANDOMIZE PEEK 23672
220 LET bo=INT (RND*ap)
230 PRINT AT 12,6;"El bote es de= ";bo
240 FOR f=1 TO 200
250 NEXT f
260 PRINT AT 20,8; FLASH 1;"PULSA UNA TECLA"
270 IF INKEY$="" THEN  GO TO 270
280 CLS
290 FOR f=1 TO 10
300    LET c(f)=30
310 NEXT f
320 GO SUB 570
330 FOR f=1 TO 19 STEP 2
340    PRINT AT f,30;CHR$ 144+CHR$ 145
350 NEXT f
360 INPUT "Por cual apuestas (1-10) ? ";nc
370 IF nc<1 OR nc>10 THEN  GO TO 360
380 PRINT AT 21,8; FLASH 1;"PULSA UNA TECLA"
390 FOR f=1 TO 200
400 NEXT f
410 IF INKEY$="" THEN  GO TO 410
420 PRINT AT 21,8;TAB 24
430 LET rn=1+INT (RND*10)
440 GO SUB 460
450 GO TO 430
455 REM
460 REM #@#@#@#@#@#@#@#@#@#@#@#
470 REM #IMPRESION DEL CABALLO#
480 REM #@#@#@#@#@#@#@#@#@#@#@#
485 REM
490 LET x=c(rn)
500 LET x=x-1
510 PRINT AT -1+(rn*2),x;CHR$ 144;CHR$ 145
520 PRINT AT -1+(rn*2),x+2;"  "
530 IF x=0 THEN  GO TO 660
540 LET c(rn)=x
550 RETURN
560 STOP
565 REM
570 REM -+-+-+-+-+-+-+-+-+-+-+-
580 REM -+-+-+- DECORADO +-+-+-
590 REM -+-+-+-+-+-+-+-+-+-+-+-
595 REM
600 LET a$="################################"
610 PRINT AT 0,0;a$;AT 20,0;a$
```

```
 620 FOR f=2 TO 18 STEP 2
 630    PRINT AT f,2;a$( TO 30)
 640 NEXT f
 650 RETURN
 660 REM * * * * * * * * * * * * *
 670 REM *    FIN DE PARTIDA     *
 680 REM * * * * * * * * * * * * *
 690 IF rn=nc THEN  GO TO 800
 700 PRINT AT 21,0; FLASH 1;"  OOOH,NO HA GANADO TU CABALLO. "
 710 FOR f=1 TO 200
 720 NEXT f
 730 PRINT AT 21,0;"                                "
 740 LET pt=pt-ap
 750 IF pt<1 THEN  PRINT AT 21,1;" * * No te queda mas dinero * *": FOR f=20 TO
1 STEP -3: BEEP .20,f: NEXT f: GO TO 920
 760 PRINT AT 21,0;"Te quedan ";pt;" pts."
 770 INPUT "Quieres seguir (s/n)? ";a$
 780 IF a$="S" OR a$="s" THEN  GO TO 170
 790 STOP
 800 PRINT AT 21,11; FLASH 1;"HAS GANADO"
 810 FOR f=-20 TO 50 STEP 4
 820    BEEP .10,f
 830 NEXT f
 840 FOR f=1 TO 100
 850 NEXT f
 860 PRINT AT 21,0;"
 870 PRINT AT 21,0;"Tienes ";pt;" pts + ";bo;" + ";ap;" = ";bo+pt+ap
 880 LET pt=bo+pt+ap
 890 INPUT "Quieres seguir jugando (s/n)? ";a$
 900 IF a$="s" OR a$="S" THEN  GO TO 170
 910 STOP
 920 PRINT AT 21,0;"                                "
 930 PRINT AT 21,0;"El caballo ganador es el ";rn;" ."
 940 INPUT "Quieres jugar otra?(s/n) ";a$
 950 IF a$="S" OR a$="s" THEN  CLS : GO TO 70
 960 STOP
 965 REM
 970 REM #####-------------#####
 980 REM #####---GRAFICOS--#####
 990 REM #####-------------#####
 995 REM
1000 FOR f=65368 TO 65383
1010 READ a
1020 POKE f,a
1030 NEXT f
1040 RETURN
1050 DATA 5,51,95,127,79,10,20,20,128,194,252,240,248,72,36,36
```




Los caballos en plena galopada.

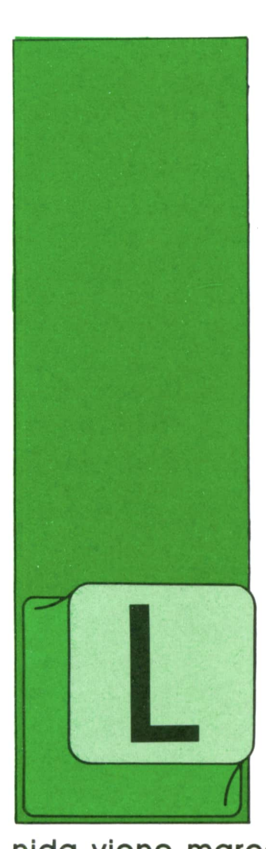




Final de la carrera.

TECNICAS DE ANALISIS

TOMA DE DATOS



LA toma de datos es una fase fundamental del procesamiento de datos, en cualquier aplicación informática. Si es cierto que la «calidad» de la información obtenida viene marcada fundamentalmente por la «calidad» de las informaciones parciales y datos que se entregan al sistema, no lo es menos que la aplicación diseñada no puede «inventar» datos y que, por tanto, es básico que los datos de entrada al sistema informático sean correctos, completos, desprovistos de errores (en la medida de lo posible) y precisos. Remedando a la expresión utilizada en los sistemas de comunicación para definir a un cierto protocolo de manejo de colas: el FIFO («First in, first out» primer dato o mensaje en la entrada, primero en la salida), usan los analistas en tono coloquial la expresión irónica BIBO («Basura» en la entrada, «basura» en la salida), para subrayar la idea comentada: los datos y resultados obtenidos por el sistema informático no pueden ser de «calidad» mayor que los datos que se le facilitan.

Y para conseguir que nuestro sistema disponga de unos datos adecuados, deben realizarse con la máxima eficacia las dos tareas básicas de la toma de datos: obtención de la información y preparación para su introducción al ordenador. Es, por tanto, de suma importancia que el documento en el que se recojan los datos de entrada esté bien diseñado para que quien deba facilitar la información lo haga eficazmente y los codificadores realicen su tarea con precisión y seguridad.

Por otro lado, incluso para el usuario individual de un ordenador personal es útil reflexionar, como hacemos a continuación, sobre cómo debe diseñar un documento que refleje los datos de entrada a un proceso, para asegurarse de que dichos datos son completos, y están puestos ordenada y cómodamente, lo que siempre ayuda a evitar errores.

En el diseño del impreso de toma de datos deberán tenerse en cuenta varios aspectos:



Aspecto general

Es importante que el aspecto general que presente el documento sea agradable. Por ello, deben evitarse los impresos con una alta densidad de preguntas por página (especialmente si cada pregunta tiene muchos datos impresos: preguntas de elección múltiple o de menú). Si es necesario, resulta preferible distribuir el cuestionario en varias páginas, antes que presentar alguna hoja muy recargada. Además, la distribución de preguntas y espacios para las respuestas debe ser equilibrada, para que resulte grata a quien debe contestar el cuestionario. Suele ser de enorme utilidad el poder introducir colores o zonas sombreadas para mejorar el aspecto general del documento. En ocasiones, incluso, puede ayudar el incluir algún diagrama o símbolo en el impreso.

Hay que tener en cuenta, además, que si el documento de toma de datos es del tipo «cuestionario de preguntas» suele llevar preimpresa mucha más información que si es del tipo «casillero de datos». En el documento con aspecto de «cuestionario» el texto de las preguntas ayuda a la correcta respuesta de datos, y permite incluir en el cuestionario comentarios o tablas auxiliares, pero resulta más «abigarrado» y, por tanto, debe cuidarse con más atención el aspecto general: inclusión de espacios vacíos, separación de zonas, sombreado de ciertas áreas, etc. Por el contrario, en el «casillero de datos», en el que se transcriben informaciones obtenidas de otros documentos o como salida de otros procesos o máquinas, están preimpresos menos textos (y más breves) y el aspecto general puede resultar más «descargado». (Normalmente, además, este tipo de documentos suele ser cumplimentado por personas más preparadas y más motivadas por esta tarea.)

En el diseño general del cuestionario hay que tener en cuenta no sólo a quien cumplimenta los datos, sino al codificador que los codifica posteriormente (si es necesaria alguna codificación de datos, como es usual) y a quien debe transcribir los datos al ordenador (grabación de datos o introducción por teclado): el documento debe ser práctico para ambos (deben encontrar las informaciones ordenadas según las van a procesar y con el resto de datos que necesitan: tablas de códigos y equivalencias, el codificador, indicaciones sobre columnas y registros, el transcriptor). En todo documento de toma de datos suelen existir dos zonas netamente separadas: el cuerpo del documento propiamente dicho (donde aparecen las preguntas y se dispone de espacio para las respuestas los datos de codificación y transcripción) y la zona de datos complementarios de identificación del documento. En ocasiones esta zona de datos complementarios (tipo de documento, nombre del mismo, datos de identificación de la aplicación a la que se destinan los datos, fecha de cumplimentación de cada fase, identificación de las personas que realizan cada tarea sobre el impreso, etc.) se descompone en dos: una a modo de encabezamiento y otra, en función de «pie» del impreso; es importante, en cualquier caso, separar nítidamente esta(s) zona(s) de informaciones complementarias (que no van a ser procesadas) de los datos propiamente dichos que constituyen el cuerpo del documento. Un buen sistema consiste en separar ambas zonas mediante distintos colores (si el coste lo permite).

Otro aspecto importante a tener en cuenta en el diseño general del documento de toma de datos es la inclusión de comentarios aclaratorios de las preguntas en sí, de tablas de códigos o, en general, de diversas instrucciones y normas útiles en la cumplimentación de los

Datos de identificación

P1 ....

Instrucciones ....

P2 ....

Instrucciones ....

P3 ....

a) anverso

P4 ....

Instrucciones ....

P5 ....

Códigos 1 2 3 4 5 6

P6 ....

Instrucciones ....

Datos finales de identificación, fechas, firmas, etc.

a) reverso

Datos de identificación

P1

P2

P3

P4

P5

P6

Datos finales

b) anverso

Instrucciones generales:

Instrucciones particulares:
P1
P2
P3
P4
P5
P6

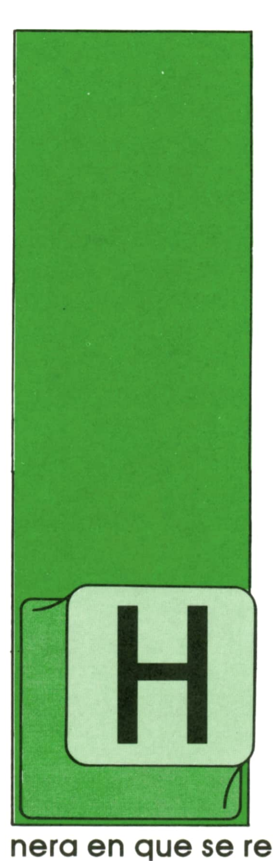
Tablas de códigos
P5

b) reverso

datos. No existe unanimidad sobre si deben ser incluidas en un solo bloque (al final del impreso o, quizá incluso, en el reverso de las hojas) o bien es preferible distribuirlas a lo largo del impreso. El procedimiento de reunir todos los datos complementarios en un solo bloque tiene la ventaja de que el documento queda más claro, queda menos «abigarrado» (lo que da menos sensación de «cansancio» a quien debe rellenarlo) y facilita la lectura de todas las instrucciones conjuntamente (lo que ayuda a una mejor comprensión global de los datos que se piden y su distribución); tiene, por el contrario, el inconveniente de que puede ser tedioso ir refiriéndose constantemente al bloque de instrucciones para responder cada pregunta (e incluso, quizá, tener que estar constantemente dando vueltas al papel) y que se presta a que la persona correspondiente acabe dejando de leer estas instrucciones.

Las ventajas e inconvenientes del sistema de «instrucciones distribuidas» son las inversas. Dependiendo del tipo de personas que va a cumplimentar el cuestionario o casillero de datos (si son profesionales de esta tarea, si tienen preparación intelectual o no, si van a tener interés en poner cuidado con las respuestas, etc.) y del propio contenido del impreso habrá que decidir si se elige un tipo de impreso u otro.

Dos modos de incluir las informaciones complementarias en un documento: «instrucciones distribuidas» (a) y «datos en bloque» (b).



TECNICAS DE PROGRAMACION

ESTRUCTURAS DE CONTROL

HASTA ahora hemos visto cómo se declara en los diferentes lenguajes la estructura y el tipo de las variables que forman parte de los programas, así como la manera en que se realizan operaciones y se asignan los valores correspondientes a dichas variables. Pero apenas hemos hablado todavía de la forma en que se dirige la ejecución de los programas, de cómo puede regularse el orden en que se van ejecutando progresivamente las instrucciones. Este orden constituye, por decirlo así, el esqueleto de los programas.

Ha llegado el momento de explicar las principales estructuras de control que pueden entrar a formar parte de los programas de ordenador en los diferentes lenguajes de programación. Entre estas estructuras vamos a describir, en primer lugar, con más detalle, tres de ellas:

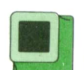
1. Bloques secuenciales.
2. Instrucciones condicionales.
3. Bucles.

Se dice que está «bien estructurado» aquel programa que no utiliza más que las tres estructuras anteriores. El arte de construir programas bien estructurados se llama «programación estructurada». En relación con esto, el conjunto de todos los programadores se divide claramente en dos grupos muy bien definidos: por un lado, los partidarios a ultranza de la programación estructurada, que se niegan a utilizar otro tipo de estructuras de control y consideran deplorable el estilo de cualquier programador que se salga de ellas; por otro lado, los partidarios de la libertad absoluta en la forma de programar, que recurren a menudo a estructuras de control más generales, como la «transferencia directa», aunque también utilizan las tres estructuras mencionadas anteriormente.

Los dos lados de la discusión tienen su parte de razón. Es cierto que los programas bien estructurados son más fáciles de revisar y corregir. Además, son más sencillos de entender por una persona diferente de su autor o por éste mismo, si ha pasado cierto tiempo desde que se construyó y se le ha olvidado su funcionamiento. Pero también es cierto que un programa estructurado suele ser más largo que un programa equivalente que utilice la transferencia directa, a menudo ocupa más espacio en memoria e incluso puede ser más lento.

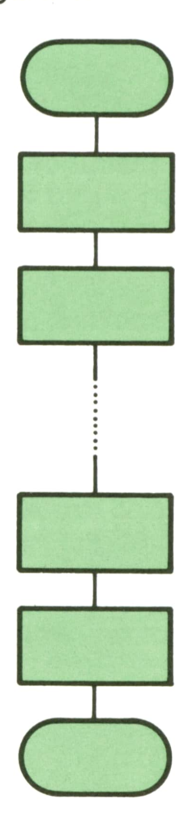
Mi opinión personal en esta cuestión se coloca más o menos en la mitad del camino entre ambos extremos, pero se inclina ligeramente hacia la libertad de programación. En efecto, creo que un programa que utiliza la instrucción de transferencia directa puede ser tan fácil de leer, interpretar y corregir como otro que prescinda de ella. Todo depende de que el programador tenga el cuidado de estructurar correctamente sus programas, sin abusar de las facilidades que le proporciona la instrucción de transferencia directa para hacerlo ilegible y enredar su funcionamiento de manera que sea casi imposible comprenderlo.

La polémica entre la programación estructurada y la programación libre se extiende a los mismos lenguajes de programación. Hay lenguajes, como el PASCAL, que han sido diseñados expresamente para que sea mucho más fácil programar estructuradamente que utilizar la transferencia directa, aunque esta estructura de control también existe en ellos. Otros lenguajes, como el BASIC, se inclinan más bien a la programación libre, dando preferencia a la transferencia directa sobre el resto de las estructuras. Por último, lenguajes como el APL se apoyan en la programación modular (que veremos más adelante) y en la potencia del lenguaje para reducir a un mínimo la necesidad de utilizar estructuras de control. Por esta razón, los organigramas de los programas APL suelen reducirse a simples secuencias de bloques.



Bloques secuenciales

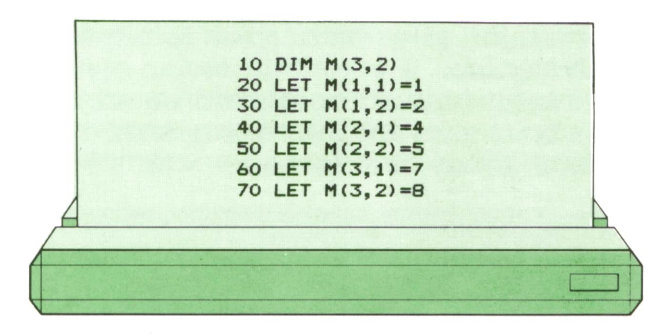
Un bloque secuencial es un conjunto de instrucciones que se ejecutan sucesivamente, una tras otra, en el mismo orden en que han sido definidas. El organigrama de un bloque secuencial se representa en la figura 1.



Cada una de las cajas rectangulares del organigrama representa una acción a ejecutar (es decir, una instrucción). Los dos bloques curvos del diagrama señalan el principio y el fin del bloque secuencial, y no corresponden a ninguna instrucción ejecutable.

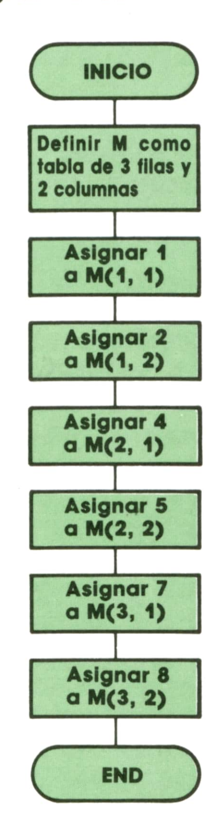
En BASIC, un bloque secuencial se puede representar de dos maneras:

1. Mediante un conjunto de instrucciones consecutivas, cada una de las cuales tiene su propio número de instrucción. Por ejemplo:

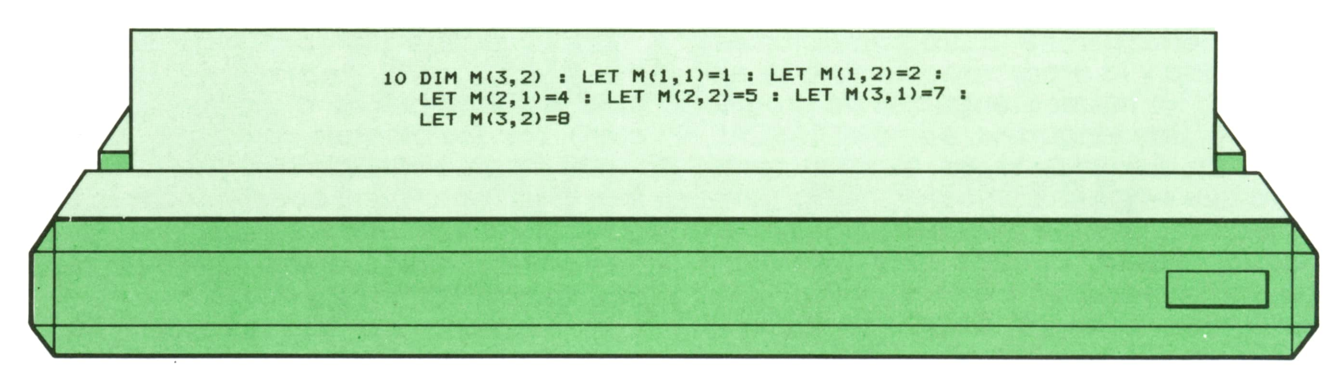


```
10 DIM M(3,2)
20 LET M(1,1)=1
30 LET M(1,2)=2
40 LET M(2,1)=4
50 LET M(2,2)=5
60 LET M(3,1)=7
70 LET M(3,2)=8
```

cuyo organigrama es:



2. Mediante varias instrucciones definidas consecutivamente en una sola línea y separadas entre sí por dos puntos. En este caso, todas ellas tendrán el mismo número de instrucción. Por ejemplo:



```
10 DIM M(3,2) : LET M(1,1)=1 : LET M(1,2)=2 :
   LET M(2,1)=4 : LET M(2,2)=5 : LET M(3,1)=7 :
   LET M(3,2)=8
```

El organigrama correspondiente a este programa es el mismo que el del programa anterior, pues ambos son totalmente equivalentes.

También puede representarse un bloque secuencial mediante una mezcla de las dos formas anteriores. Por ejemplo:

```
10 DIM M(3,2)
20 LET M(1,1)=1 : LET M(1,2)=2
30 LET M(2,1)=4 : LET M(2,2)=5
40 LET M(3,1)=7 : LET M(3,2)=8
```

A veces, esta última forma de representación (a la que también corresponde el mismo organigrama) es la más conveniente. En el ejemplo dado, hemos separado la declaración de la tabla M de las asignaciones de valor a cada una de las filas, lo que hace más condensado el bloque completo (que pasa a tener cuatro instrucciones numeradas en lugar de siete) sin hacerlo más difícil de entender, lo que puede ocurrir si se abusa de líneas demasiado largas, como en el segundo ejemplo.

En PASCAL, un bloque secuencial empieza siempre por la palabra reservada BEGIN, que indica el principio del bloque, y termina en la palabra reservada END, que señala su fin. Si nos fijamos en los organigramas que representan bloques secuenciales, podemos pensar que BEGIN y END son las instrucciones de PASCAL que corresponden a los dos bloques curvos que se colocan siempre al principio y al final de las estructuras de este tipo.

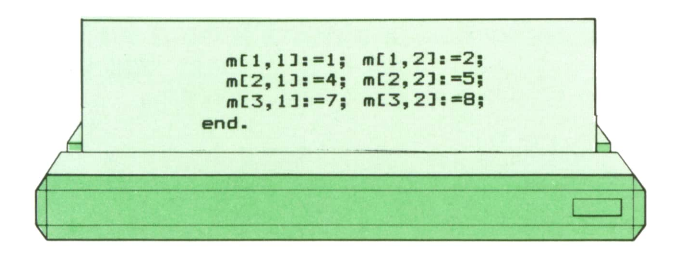
Veamos un ejemplo, que corresponde a la programación en PASCAL del mismo ejemplo dado anteriormente en BASIC, y que, por tanto, tendrá el mismo organigrama que aquél:

```
program TABLA;
var
  m: array[1..3,1..2] of integer;
begin
  m[1,1]:=1;
  m[1,2]:=2;
  m[2,1]:=4;
  m[2,2]:=5;
  m[3,1]:=7;
  m[3,2]:=8;
end.
```

Se observará que todo programa PASCAL comienza por una instrucción que define su nombre («programa TABLA», en nuestro ejemplo) y se divide en dos partes principales: un conjunto de declaraciones (que en este caso comienza por la palabra reservada «var») y un conjunto de instrucciones ejecutables, que, en general, formarán un bloque secuencial y que, por tanto, deben comenzar por la palabra reservada «begin» y terminar con «end».

En PASCAL las instrucciones han de terminar siempre en punto y coma. Por esta razón, su principio y su fin queda claramente delimitado, por lo que no se ponen restricciones a la forma en que tienen que escribirse: se pueden mezclar varias instrucciones en la misma línea, dejar líneas en blanco, partir una línea en dos, encolumnar ciertas instrucciones de una manera y otras de otra, etc. Se entiende que el programador deberá seleccionar siempre aquella disposición que más contribuya a la legibilidad del programa, es decir, a hacerlo más comprensible. Veamos, como ejemplo, otra forma en que podría escribirse el programa anterior:

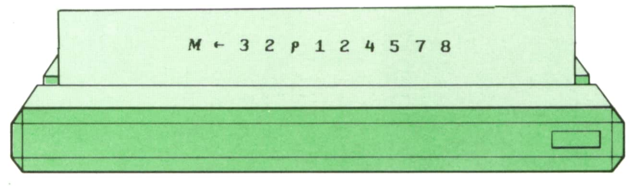
```
program TABLA;
var
  m: array[1..3,1..2] of integer;
begin
```



```
m[1,1]:=1; m[1,2]:=2;
m[2,1]:=4; m[2,2]:=5;
m[3,1]:=7; m[3,2]:=8;
end.
```

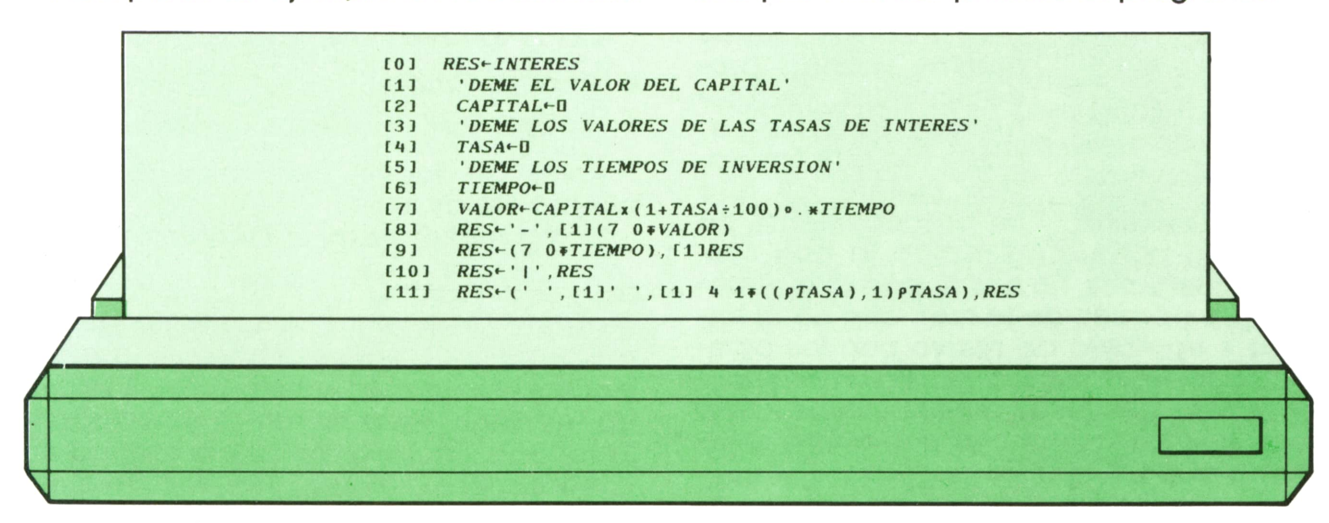
En el lenguaje APL, un bloque secuencial se representa, como en BASIC, con un conjunto de líneas consecutivas numeradas adecuadamente. Algunos intérpretes de APL permiten unir varias instrucciones en una sola línea, separándolas con un símbolo parecido al diamante de la baraja inglesa (♦), pero esto no es un elemento estándar del lenguaje y hay otros intérpretes que no lo permiten.

Para poner un ejemplo en APL tenemos que buscar un caso más complicado, ya que el programa que hemos estado siguiendo en BASIC y PASCAL se reduce en APL a una sola instrucción, que estrictamente hablando no constituye un bloque secuencial:



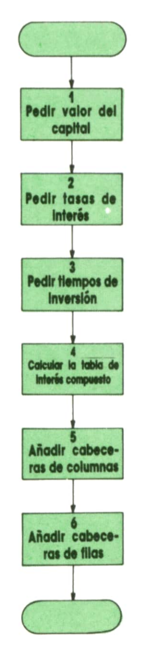
```
M ← 3 2 ⍴ 1 2 4 5 7 8
```

Consideremos, por tanto, el ejemplo que ya vimos en el capítulo primero, que permitía obtener la tabla de interés compuesto correspondiente a un capital determinado invertido con diferentes tipos de interés y durante diversos períodos de tiempo. Veamos primero el programa:

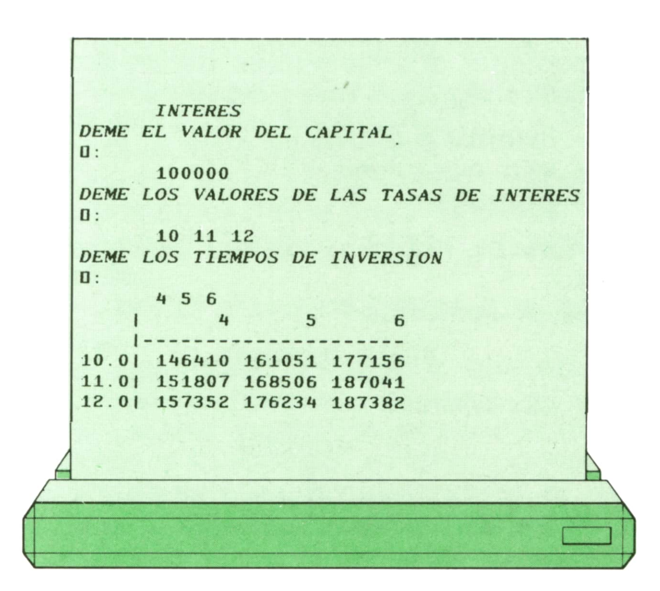


```
[0]  RES←INTERES
[1]   'DEME EL VALOR DEL CAPITAL'
[2]   CAPITAL←⎕
[3]   'DEME LOS VALORES DE LAS TASAS DE INTERES'
[4]   TASA←⎕
[5]   'DEME LOS TIEMPOS DE INVERSION'
[6]   TIEMPO←⎕
[7]   VALOR←CAPITAL×(1+TASA÷100)∘.*TIEMPO
[8]   RES←'-',[1](7 0⍕VALOR)
[9]   RES←(7 0⍕TIEMPO),[1]RES
[10]  RES←'|',RES
[11]  RES←(' ',[1]' ',[1] 4 1⍕((⍴TASA),1)⍴TASA),RES
```

cuyo organigrama es:

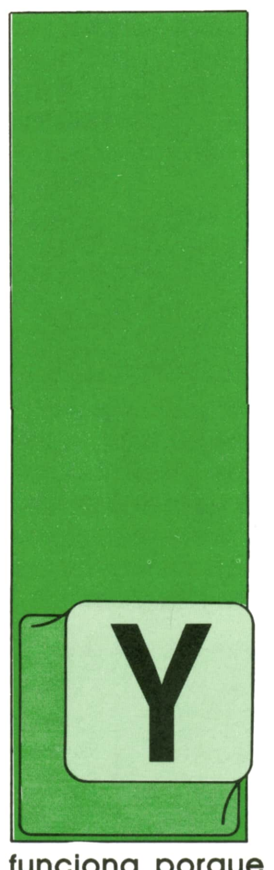


y cuya ejecución da el siguiente resultado:



```
      INTERES
DEME EL VALOR DEL CAPITAL
⎕:
      100000
DEME LOS VALORES DE LAS TASAS DE INTERES
⎕:
      10 11 12
DEME LOS TIEMPOS DE INVERSION
⎕:
      4 5 6
    |     4      5      6
    |---------------------
10.0| 146410 161051 177156
11.0| 151807 168506 187041
12.0| 157352 176234 187382
```

LOGO



Cómo corregir los procedimientos

Y A hemos podido hacer muchas cosas nuevas usando procedimientos. Pero tenemos una pega. Cada vez que nos damos cuenta de que un procedimiento no funciona porque no realiza lo que nosotros queremos, no nos queda más remedio que borrarlo de la memoria de la tortuga y escribirlo de nuevo con los cambios que sean necesarios. Pero lo normal no es que todo el procedimiento esté mal, sino que sólo haya que cambiar algunas partes y el resto siga igual. Por ello, el borrar todo un procedimiento para cambiar unos cuantos comandos es un poco absurdo.

Para evitar esto existe un comando que permite obtener la definición de uno o varios procedimientos para modificarla. El modificar una definición de un procedimiento significa que podemos:

— insertar o añadir
— eliminar o borrar
— cambiar

comandos, números, letras..., dentro de ella.

Así, el comando

EDITA "nombre

o en abreviatura

ED "nombre

nos muestra en la pantalla la definición del procedimiento **nombre** para que podamos cambiar aquello que esté mal.

En lugar de ver la definición de un solo procedimiento, nos puede interesar el visualizar varios a la vez. Para ello, tendremos que poner los nombres de los distintos procedimientos en una lista. Así, nos quedaría

EDITA (nombre1 nombre2 ... nombren)

o en abreviatura

ED (nombre1 nombre2 ... nombren)

Antes de seguir, hemos de tener en cuenta varias cosas.

Como hemos visto, si escribimos

ED "nombre

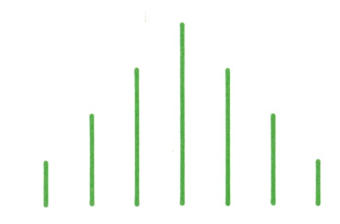
obtenemos en pantalla la definición del procedimiento llamado **nombre** siempre y cuando este procedimiento esté definido, es decir, se lo hayamos enseñado a la tortuga. En caso de que la tortuga no conozca ese nombre, nos mostrará en pantalla una sola línea que contendrá

PARA nombre

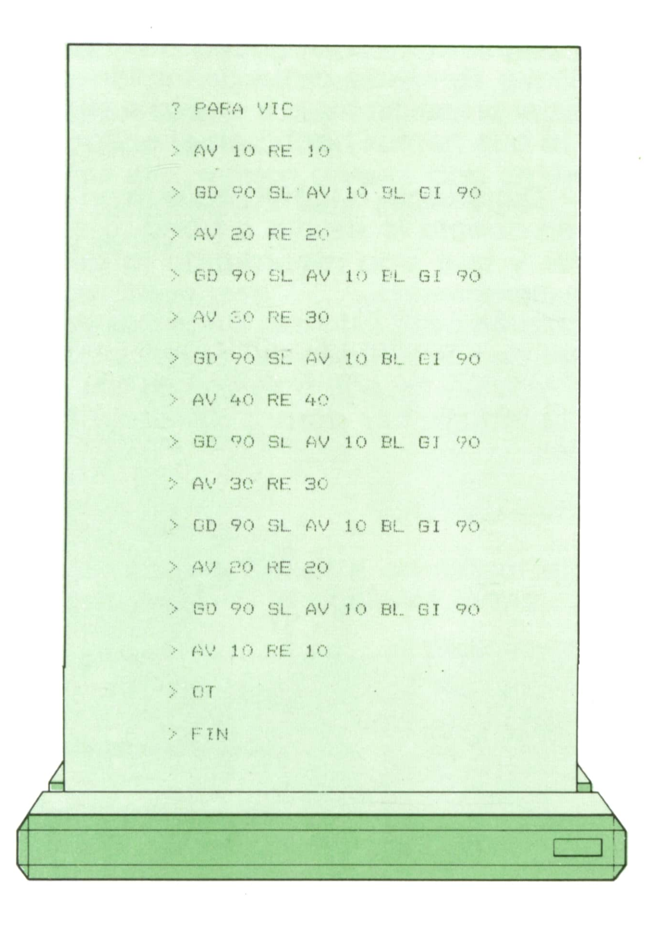
y, por tanto, tendremos la posibilidad de definir el procedimiento. Es decir, es otra manera de enseñar cosas nuevas a la tortuga.

Por último, si ponemos el comando ED sin ningún nombre, se nos mostrará el último procedimiento que hayamos visualizado mediante este comando.

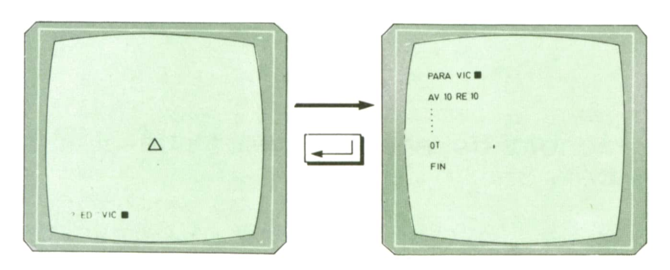
Veamos todo esto con un ejemplo. Supongamos que le hemos enseñado a la tortuga un procedimiento llamado VIC que hace la siguiente figura:



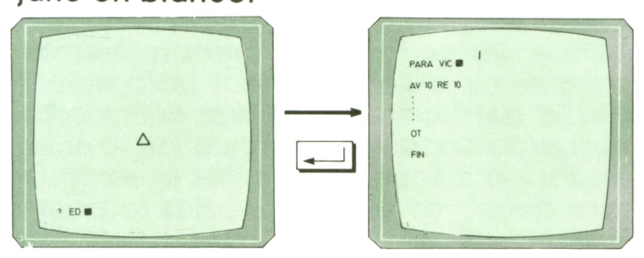
Este procedimiento lo habremos definido así:



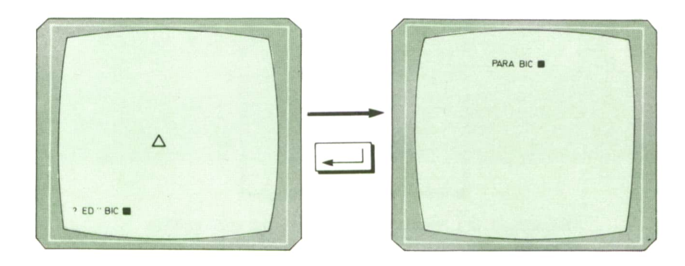
Si ahora queremos ver este procedimiento para modificar algo, escribiremos



Si salimos del editor y queremos volver a visualizar el mismo procedimiento, en lugar de escribir su nombre podemos dejarlo en blanco:



Por último, si de nuevo queremos ver este procedimiento y tecleamos mal:



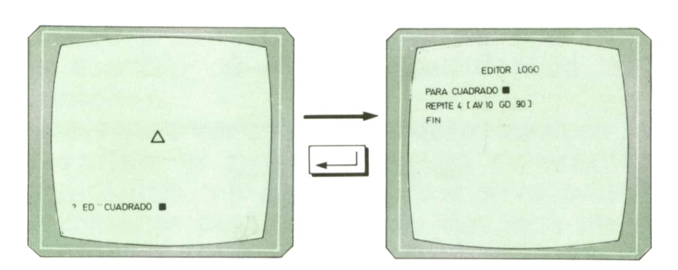
la tortuga no conoce ese nombre y nos da la posibilidad de que le enseñemos lo que significa.



Funcionamiento del editor

Cuando le indicamos a la tortuga que queremos que ejecute el comando ED, decimos que hemos entrado en el **editor.**

Es fácil darse cuenta de cuándo se está en el editor, debido a que la pantalla aparece en modo texto (no hay parte de dibujos) y, además, se nos muestra un mensaje en la parte de arriba o de abajo que lo indica.



También se distingue porque delante de las líneas donde están escritos o se escriben los comandos no aparece ningún carácter especial (?, >) como ocurre el resto de las veces.

Cuando en la pantalla tenemos el editor, podemos hacer muchas cosas:

— añadir partes nuevas al procedimiento;

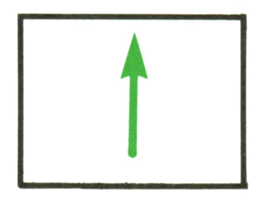
— borrar aquellas partes que no sirven;

— cambiar o corregir lo que no esté completamente bien;

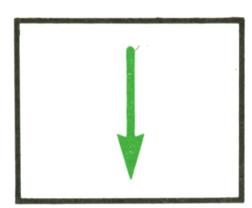
— incluso podemos enseñarle cosas nuevas a la tortuga, es decir, definir un procedimiento que no existiera.

Para poder realizar estas funciones, necesitamos tener unas teclas que nos sir-

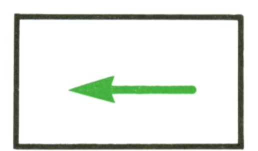
van para movernos por la definición, borrar caracteres, añadir líneas... Las más importantes son las siguientes:



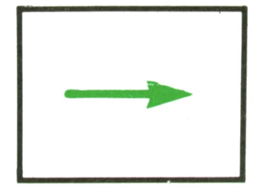
Mueve el cursor a la línea anterior.



Mueve el cursor una línea abajo.



Mueve el cursor un carácter a la izquierda.



Mueve el cursor un carácter a la derecha.



Borra el carácter que está a la izquierda del cursor.



Crea una línea en blanco debajo de aquella en la que estaba situado el cursor.

En general, estas teclas y su función dependen del ordenador que se utilice.

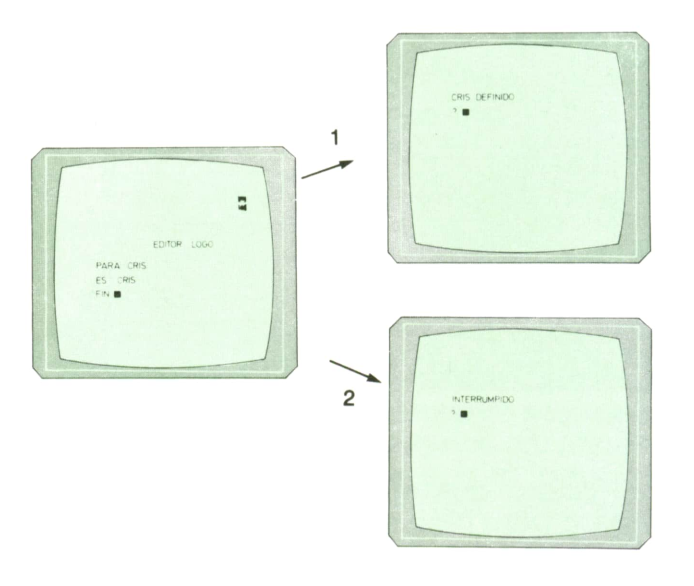
Una vez que hemos corregido nuestro procedimiento, tenemos dos formas de salir del editor:

1. Diciéndole a la tortuga que queremos que se olvide de la definición antigua del procedimiento y que la sustituya por lo que hemos hecho en el editor.

2. Diciéndole a la tortuga que no tenga en cuenta lo escrito o borrado en el editor y que siga recordando la definición de antes.

También para esto existen teclas especiales que dependen del ordenador.

En función de cómo salgamos del editor, la tortuga nos dará un mensaje diferente:

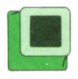


1. La tortuga considera la definición nueva.

2. La tortuga considera la definición antigua.

Una vez que hemos modificado un procedimiento que estaba mal, podemos volver a probarlo para ver si funciona. En caso de que no sea así, podemos usar de nuevo el editor para irlo cambiando hasta que realice lo que nosotros queremos.

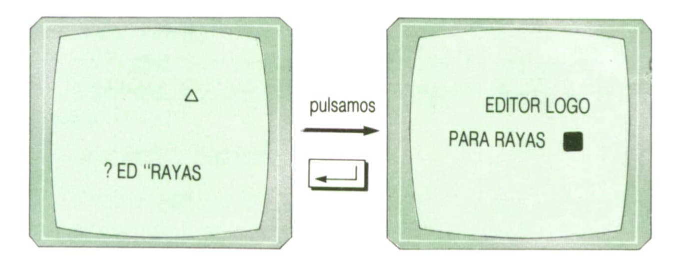
Antes de utilizar el editor para modificar la definición de tus procedimientos, recuerda consultar el manual de tu ordenador para saber qué teclas puedes utilizar dentro de él y qué teclas te permiten salir de él.



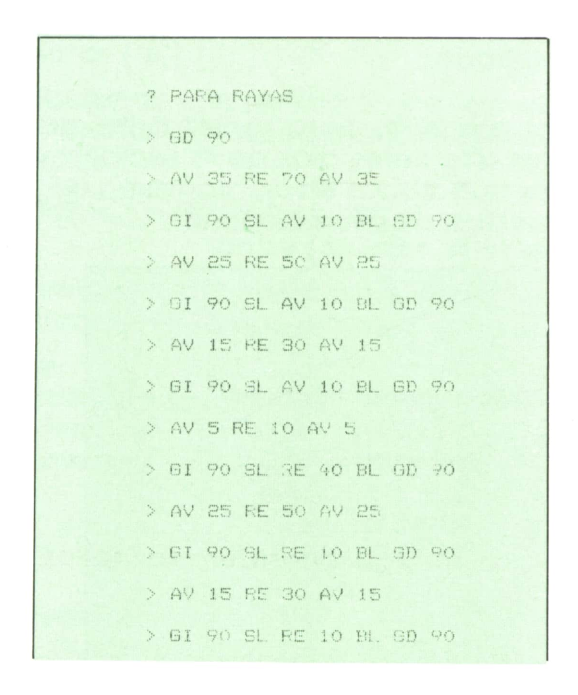
Dos formas de definir procedimientos

Vamos a dar un repaso a las dos maneras que tenemos de definir procedimientos, es decir, de enseñar a la tortuga cosas nuevas.

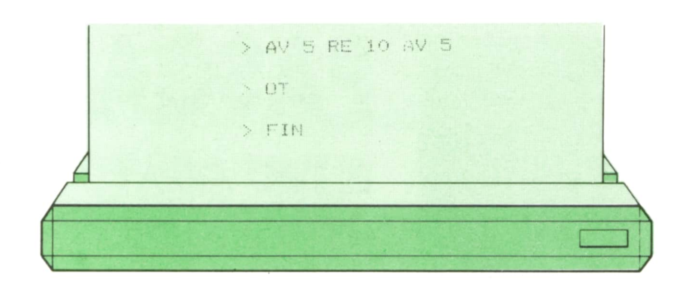
Para ello, vamos a verlo con un ejemplo. Supongamos que queremos hacer esta figura:



Una manera consiste en utilizar el comando PARA de la siguiente forma:



```
? PARA RAYAS
> GD 90
> AV 35 RE 70 AV 35
> GI 90 SL AV 10 BL GD 90
> AV 25 RE 50 AV 25
> GI 90 SL AV 10 BL GD 90
> AV 15 RE 30 AV 15
> GI 90 SL AV 10 BL GD 90
> AV 5 RE 10 AV 5
> GI 90 SL RE 40 BL GD 90
> AV 25 RE 50 AV 25
> GI 90 SL RE 10 BL GD 90
> AV 15 RE 30 AV 15
> GI 90 SL RE 10 BL GD 90
```



```
> AV 5 RE 10 AV 5
> OT
> FIN
```

Con este comando tenemos la ventaja de que en cuanto escribamos FIN y pulsemos RETURN podemos ejecutar el procedimiento, pero tiene el inconveniente de que tendremos que entrar en el editor si nos damos cuenta de que hemos escrito algo mal, ya que con el comando PARA no nos podemos mover por la definición de un procedimiento ni corregirla.

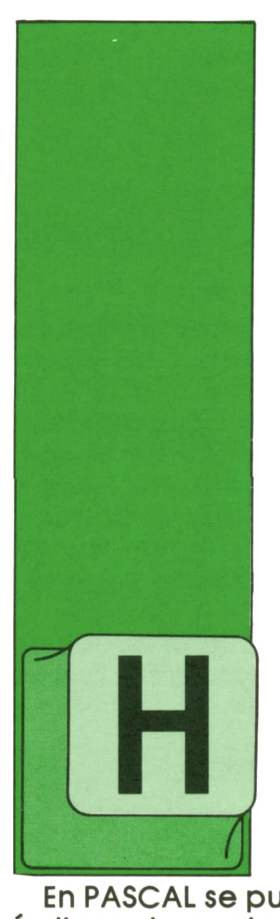
Otra manera de definir este procedimiento es usando el comando ED

y escribiendo en el editor la definición anterior.

En caso de que nos equivoquemos al escribir el procedimiento podremos corregirlo, ya que el editor sirve principalmente para eso, pero hay que tener cuidado cuando salgamos de él. Debemos pulsar la tecla que haga que la tortuga guarde en su memoria lo que acabamos de escribir, porque si no, no sabrá lo que tiene que hacer cuando le digamos que ejecute el procedimiento.

PASCAL

EL TIPO ARRAY



HASTA ahora sólo hemos utilizado como índices números pertenecientes a subrangos del tipo INTEGER; sin embargo, no es éste el único tipo permitido.

En PASCAL se pueden definir tablas con índices de cualquier tipo escalar o subrango, siempre que el espacio de memoria necesario para la tabla resultante esté dentro de los límites impuestos por la memoria del ordenador y el propio compilador.

Por ejemplo, para guardar los gastos de los diferentes días de la semana y las horas trabajadas en un programa de presupuestos, podríamos poner:

```
program Presupuesto;

type
  DiaDeLaSemana_t =
      (Lunes,Martes,Miercoles,Jueves,Viernes,Sabado,Domingo);
  DiaLaborable_t  = Lunes..Viernes;
  Gastos_t        = array [DiaDeLaSemana_t] of real;
  Horas_t         = array [DiaLaborable_t] of integer;

var
  Gastos: Gastos_t;
  Horas : Horas_t;
  Dia   : DiaDeLaSemana_t;

begin
 ...

 Horas [Martes]:= 5;
 for Dia:= Lunes to Domingo do
    writeln (Gastos [Dia]);
 ...

end.
```

Los elementos constitutivos de una variable del tipo array pueden ser de CUALQUIER tipo que se nos ocurra; tipos predefinidos, tipos nuevos e incluso tablas de otro tipo.

Imaginemos que en el colegio en que

se va a utilizar el programa de medias que hicimos anteriormente haya varios grupos llamados A, B, C, D y E. Una vez más, podríamos crear para cada uno una variable tipo array diferente, Notas-A, Notas-B, etc., pero entonces habría que repetir las instrucciones para cada una.

En lugar de eso, podríamos crear una tabla de... tablas de notas:

```
type
   Alumno_t = 1..100;
   Grupo_t = array (Alumno_t) of real;
   Notas_t = array ('A'..'E') of Grupo_t;
var
   Notas : Notas_t;
```

Para referirnos entonces a la tabla de notas del grupo B pondríamos:

```
   Notas ('B')
```

y para referirnos a la nota del alumno 37 del grupo B haríamos, por tanto:

```
   Notas ('B') (37)
```

Haciendo una definición de tipo «sobre la marcha», se puede evitar definir el tipo Grupo-t:

```
   Notas_t = array ('A'..'E') of
                  array (Alumno_t) of real;
```

Teniendo en cuenta las diferentes asignaturas, podríamos tener:

```
type
   Materias_t = (Física, Lengua, Inglés);
   Alumno_t   = 1..100;
   Notas_t    = array (Materias_t) of
                  array ('A'..'E') of
                     array (Alumno_t) of
                     real;
var
   Notas: Notas_t;
```

y tendríamos entonces, por ejemplo:

```
Notas (Fisica) ('B') (37)
```

que sería la nota del alumno 37 del grupo B en física. A este tipo de variables se les llama tablas (o arrays) multidimensionales.

Como se ve, el total de datos que se podría guardar en Notas sería 3 asignaturas por 5 grupos por 100 alumnos igual a 1.500 números; por tanto, debe haber suficiente espacio en memoria para albergar, además del programa y las otras variables, los 1.500 números reales (normalmente, y dependiendo del compilador, un número de tipo Real ocupa entre 5 y 8 bytes).

En PASCAL la expresiones como:

```
array (Materias_t) of
   array ('A'..'E') of
      array (Alumno_t) of real
```

y

```
   Notas (Fisica) ('B') (37)
```

se pueden abreviar así:

```
   array (Materias_t, 'A'..'E', Alumno_t) of
real
   Notas (Fisica, 'B', 37)
```

es decir, poniendo los diferentes subíndices entre una única pareja de corchetes, y separándolos con comas.

Por supuesto, el tipo tabla es perfectamente válido a la hora de definir variables locales de procedimientos o de pasar parámetros a éstos. Conviene recordar que el paso de parámetros por valor, al implicar un proceso de copia de datos, puede ser más lento que el paso de parámetros por nombre si la tabla ocupa mucho espacio de memoria.

A propósito de lentitudes, la utilización de tablas en lugar de muchas variables simples tiene un inconveniente que normalmente es despreciable frente a las ventajas que depara: cuando se hace referencia a un elemento de una tabla, por lo menos hay que obtener dos datos de la memoria del ordenador, primero el índice y luego el propio elemento; sin embargo, con las variables simples, se accede directamente al dato de interés y por ello el manejo de tablas es un poco más lento.

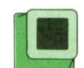


Tabla de caracteres

Un caso especial de array es aquél cuyos elementos son caracteres: al ser tablas de éstos, permiten guardar textos.

Supongamos que tenemos definida la variable Texto como array (1..4) of char; con lo que sabemos, para guardar la palabra 'Luis' haríamos:

```
   Texto (1) := 'L';
   Texto (2) := 'u';
   Texto (3) := 'i';
   Texto (4) := 's';
```

Sin embargo, la mayoría de los compiladores permiten poner:

Texto := 'Luis';
(* Deben ser exactamente 4 caracteres *)

También suele ser posible escribir o leer con una sola instrucción estas tablas:

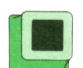
readln (Texto);
writeln (Texto);

Esta última instrucción, por ejemplo, equivaldría a:

for Indice := 1 to 4 do
write (Texto [Indice]);
writeln;

por otra parte, la instrucción de lectura tomaría sólo los primeros cuatro caracteres si hubiésemos tecleado de más, y rellenaría con espacios en blanco o con el carácter cuyo ordinal es 0 (chr (0)), según el compilador, los que faltasen hasta cuatro si hubiéramos tecleado de menos antes de pulsar la tecla de RETURN (o INTRO...).

A las tablas de caracteres se les llama habitualmente «STRINGS» (cuerda en inglés). No obstante, y aunque en principio, las tablas de caracteres son las adecuadas para guardar textos, algunos compiladores disponen para esto de un tipo predefinido especial llamado precisamente string que facilita algunas cosas y que es compatible sólo en parte con las tablas. Quizá en otro momento volvamos sobre esto pero, en general, utilizaremos en todos los programas que escribamos exclusivamente tablas de caracteres a la hora de manipular textos.



Un ejemplo

Vamos a ver cómo podríamos guardar en memoria la situación del tablero del contrario en una partida de batalla naval (o de «barquitos»).

En cada posición del tablero puede haber cuatro situaciones: que haya agua, que haya un barco tocado, que haya un barco hundido o que no se sepa lo que hay; podríamos así definir por enumeración el tipo de los posibles valores que puede tomar una posición, al que llamaríamos, por ejemplo, Estado-t.

Para almacenar la situación de una fila del tablero tendríamos entonces un array (1..10) of Estado-t y, por tanto, el tablero completo quedaría como una tabla de filas, o sea:

```
program Barquitos;

type
  Estado_t   = (Agua, Tocado, Hundido, NiIdea);
  Fila_t     = 'A'..'J';
  Columna_t  =  1..10;
  Tablero_t  = array [Fila_t] of array [Columna_t] of Estado_t;

  (* o bien = array [Fila_t, Columna_t] of Estado_t; *)

var
  Propio,
  Contrario: Tablero_t;
  ...
```

entonces podríamos escribir cosas como:

Contrario ('B', 7) := Agua;
(* o sea, en B7 hay agua *)
if Contrario ('J', 9) = Ni Idea then...

Si quisiéramos visualizar en pantalla la situación de un tablero escribiendo, por ejemplo, un espacio en blanco en las casillas desconocidas, una T en donde haya barco tocado, una H en los hundidos y una A donde se sepa que hay agua, podríamos escribir:

```
procedure MuestraTablero (T: Tablero_t);

 var
  Fila   : Fila_t;
  Columna: Columna_t;

 begin
  for Fila:= 'A' to 'J' do  (* Dibujar de fila en fila: *)
    begin
      (* Primero pintar la fila de columna en columna: *)
      for Columna:= 1 to 10 do
        begin
         if T [Fila,Columna] = NiIdea  then write (' ');
         if T [Fila,Columna] = Tocado  then write ('T');
         if T [Fila,Columna] = Hundido then write ('H');
         if T [FILa,Columna] = Agua    then write ('A')
        end;
       (* Tras acabar cada fila, bajar una línea *)
      writeln
    end
 end;
```

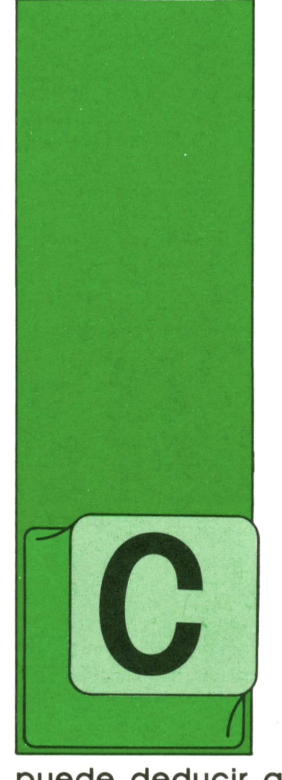
Para presentar los tableros bastaría entonces con escribir:

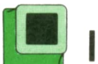
MuestraTablero (Propio);
MuestraTablero (Contrario);

El PASCAL permite así definir estructuras de datos sumamente adaptadas al problema real que se desee resolver. Esto redunda en una mayor claridad de los programas y por tanto en una programación más eficiente y depurada con menor cantidad de errores.

OTROS LENGUAJES

COBOL





Introducción

COBOL es la abreviatura de «Common Business Oriented Language», que significa lenguaje orientado normalmente a los negocios.

De su nombre se puede deducir que se trata de un lenguaje de programación dedicado principalmente al desarrollo de aplicaciones de gestión y, por tanto, es uno de los lenguajes más extendidos actualmente.

Es el resultado de una reunión celebrada en Estados Unidos en 1959, entre fabricantes de ordenadores y usuarios de los mismos.

Un año después, aparece la primera versión del COBOL, llamada COBOL-60, que fue depurada progresivamente, dando lugar a sucesivas versiones: COBOL-61, COBOL-62, ...

En 1968 se dan los primeros pasos para normalizar el lenguaje.

Se ocupaba de ello el American National Standard Institute. De esta forma surgió el ANSI COBOL.

El COBOL resulta ser un lenguaje fácilmente legible. Si los programas están bien estructurados, resulta fácil entenderlos, ya que la sintaxis del lenguaje es muy similar a la gramática inglesa.

Es un lenguaje orientado, como ya se ha dicho antes, a las aplicaciones de gestión por su estructura y diseño; en cambio, no es adecuado para realizar programas que requieran cálculos de expresiones matemáticas complejas.

Los programas que aquí se presentan se han realizado con un IBM PC y COBOL MICROSOFT versión 2.00.



Estructura

Un programa COBOL tiene una estructura rígida. Está formado por cuatro «divisiones», que a su vez se subdividen en «secciones». El nombre de estas divisiones es:

— Identification division.
— Environment division.
— Data division.
— Procedure division.

En la primera (Identification), se da nombre al programa y se proporciona al compilador una información adicional: nombre del autor, fecha en la que se escribió, fecha de la última compilación, mensajes de seguridad y comentarios.

De esta división sólo es obligatorio el nombre del programa. El resto de las cláusulas son opcionales.

En la segunda división (Environment) se enumeran los ficheros (tanto en cintas, discos, como impresoras) que utiliza el programa, asignándolos a determinados periféricos y reseñando su organización y modo de acceso.

En la Data division se describen los registros de los ficheros que han sido declarados en la anterior división.

El conjunto de sentencias que realmente se van a ejecutar se encuentran en la Procedure division. Se puede decir que ésta es la parte «ejecutable», siendo las demás de «declaración y definición».



Normas de escritura

A diferencia de la gran mayoría de lenguajes, un programa COBOL debe ajus-

tarse a unas normas estrictas de escritura.

Para explicar estas reglas se va a utilizar la hoja de codificación.

Cada línea de la hoja de codificación se corresponde con una línea de la pantalla.

Las seis primeras columnas se utilizan para numerar las diferentes líneas del programa. Estas eran útiles cuando se trabajaba con fichas perforadas; actualmente, al usar terminales, esas columnas se dejan en blanco.

En la columna 7 pueden aparecer tres caracteres, cada uno de ellos tiene un significado diferente.

— Espacio en blanco: Línea normal.
— '*': Línea de comentario.
— '-': Indica la continuación de la línea anterior.

Las columnas comprendidas entre la 8 y la 11 (ambas inclusive) se conocen como *margen A*.

El *margen B* está formado por las columnas que van desde la 12 hasta la 72.

En los márgenes A y B deben comenzar determinadas claúsulas. La colocación de cada una se irá determinando a medida que aparezcan en el texto.

El nombre de las cuatro divisiones debe comenzar en el margen A.

Además de las normas de situación, existen tambien normas de puntuación que pueden resumirse en los siguientes puntos:

— Los caracteres coma, punto y punto y coma no pueden ir precedidos de espacio, pero deben ir seguidos al menos de uno.
— El paréntesis izquierdo no puede ir seguido de un espacio.
— El paréntesis derecho no puede ir precedido de un espacio.
— El signo igual y los operadores aritméticos deben estar precedidos y seguidos por un blanco.

A continuación se expone un programa COBOL, que muestra la estructura del mismo, así como la colocación de las instrucciones en sus márgenes correspondientes y las distintas normas de puntuación.

A medida que se avance en el texto se irá entendiendo cada componente del programa.

```
       IDENTIFICATION DIVISION.
        PROGRAM-ID. LISTADO.
        AUTHOR. E.P.H.
        DATE-WRITTEN. ABRIL-87.
        DATE-COMPILED.

      *  Programa que obtiene un listado del personal a partir de un
      * fichero de empleados.

       ENVIRONMENT DIVISION.

        CONFIGURATION SECTION.

        INPUT-OUTPUT SECTION.

          FILE-CONTROL.

              SELECT PERSONAL ASSIGN TO DISK
                              ORGANIZATION LINE SEQUENTIAL.

              SELECT IMPRESO  ASSIGN TO PRINTER.
       DATA DIVISION.

        FILE SECTION.

         FD PERSONAL  LABEL RECORD STANDARD
                      VALUE OF FILE-ID 'PERSONAL.DAT'.

         01 REG-PERSONAL.
            05 COD-EMPLEADO              PIC 9(5).
            05 NOM-EMPLEADO.
               10 APELLIDO-1             PIC X(15).
               10 APELLIDO-2             PIC X(15).
               10 NOMBRE                 PIC X(12).
            05 DIRECCION.
               10 CALLE                  PIC X(23).
               10 COD-POSTAL             PIC X(5).
            05 TELEFONO                  PIC X(11).
```

```
FD IMPRESO   LABEL RECORD OMITTED.

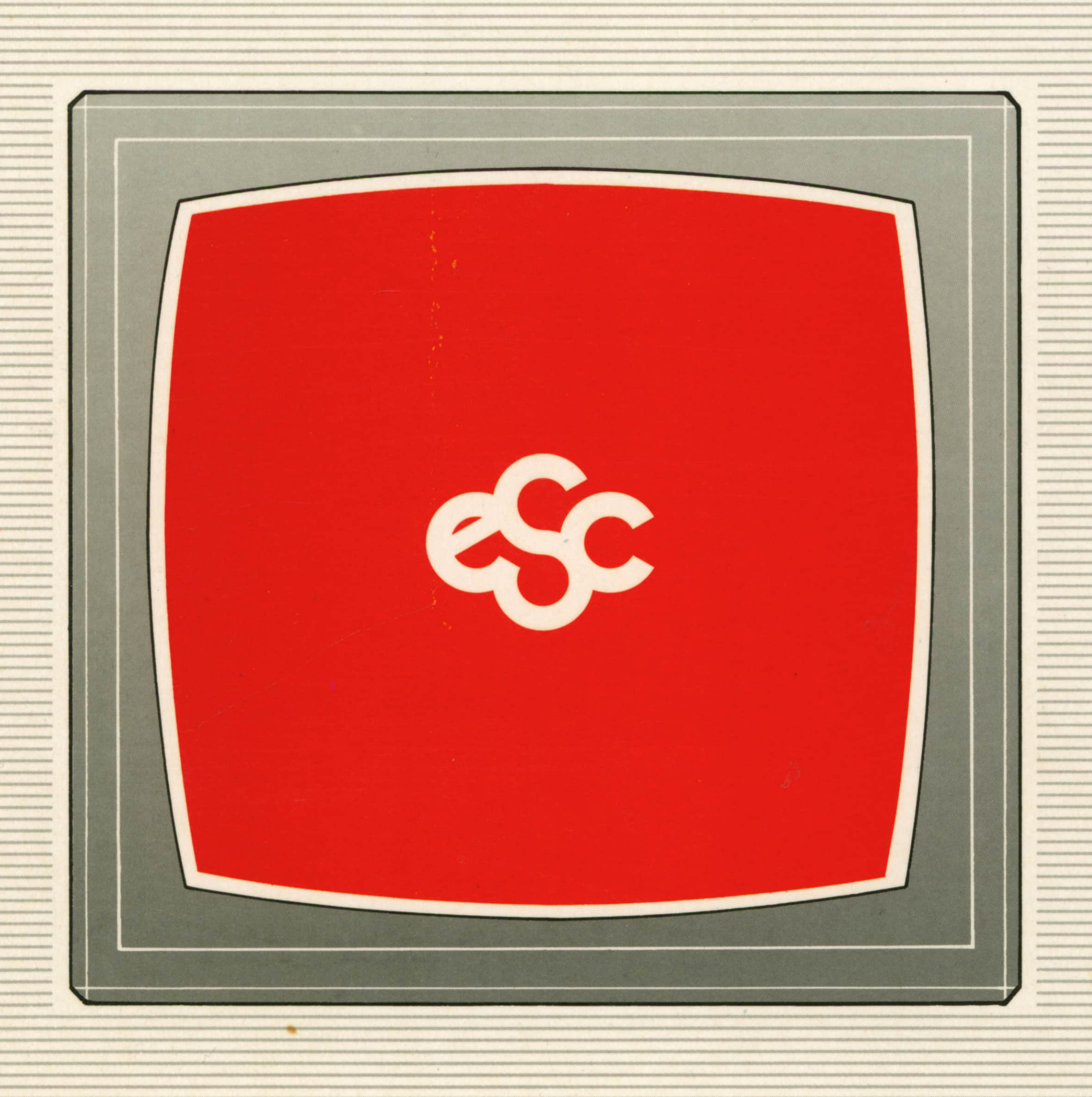
01 LINEA                               PIC X(80).
WORKING-STORAGE SECTION.

 01 FINAL-FICHERO                      PIC X           VALUE 'N'.

 01 CABECERA1.
     05 FILLER                         PIC X(20)       VALUE SPACES.
     05 FILLER                         PIC X(19)       VALUE
             'LISTADO DE PERSONAL'.
 01 SUBRAYAR1.
     05 FILLER                         PIC X(20)       VALUE SPACES.
     05 FILLER                         PIC X(19)       VALUE ALL '-'.
 01 CABECERA2.
     05 FILLER                         PIC X(5)        VALUE SPACES.
     05 FILLER                         PIC X(6)        VALUE 'CODIGO'.
     05 FILLER                         PIC X(17)       VALUE SPACES.
     05 FILLER                         PIC X(18)       VALUE
             'APELLIDOS Y NOMBRE'.
     05 FILLER                         PIC X(19)       VALUE SPACES.
     05 FILLER                         PIC X(8)        VALUE
             'TELEFONO'.
 01 SUBRAYAR2.
     05 FILLER                         PIC X(5)        VALUE SPACES.
     05 FILLER                         PIC X(6)        VALUE ALL '-'.
     05 FILLER                         PIC X(4)        VALUE SPACES.
     05 FILLER                         PIC X(45)       VALUE ALL '-'.
     05 FILLER                         PIC X(3)        VALUE SPACES.
     05 FILLER                         PIC X(11)       VALUE ALL '-'.
 01 DETALLE.
     05 FILLER                         PIC X(6)        VALUE SPACES.
     05 COD-EMPLEADO-SAL               PIC 9(5).
     05 FILLER                         PIC X(4)        VALUE SPACES.
     05 APELLIDO1-SAL                  PIC X(15).
     05 FILLER                         PIC X           VALUE SPACES.
     05 APELLIDO2-SAL                  PIC X(15).
     05 FILLER                         PIC XX          VALUE ', '.
     05 NOMBRE-SAL                     PIC X(12).
     05 FILLER                         PIC X(3)        VALUE SPACES.
     05 TELEFONO-SAL                   PIC X(11).

 01 CONT-LINEAS                        PIC 99          VALUE 51.

PROCEDURE DIVISION.
PRINCIPAL.
        OPEN INPUT  PERSONAL.
        OPEN OUTPUT IMPRESO.
        PERFORM LEER-PERSONAL.
        PERFORM ESCRIBIR UNTIL FINAL-FICHERO = 'S'.
        CLOSE PERSONAL
               IMPRESO.
        STOP RUN.
LEER-PERSONAL.
        READ PERSONAL AT END MOVE 'S' TO FINAL-FICHERO.
ESCRIBIR.
        MOVE APELLIDO-1    TO APELLIDO1-SAL.
        MOVE APELLIDO-2    TO APELLIDO2-SAL.
        MOVE NOMBRE        TO NOMBRE-SAL.
        MOVE COD-EMPLEADO TO COD-EMPLEADO-SAL.
        MOVE TELEFONO      TO TELEFONO-SAL.
        IF CONT-LINEAS > 50
           PERFORM ESCRIBIR-CABECERAS.
        WRITE LINEA FROM DETALLE AFTER 1.
        ADD 1 TO CONT-LINEAS.
        PERFORM LEER-PERSONAL.
ESCRIBIR-CABECERAS.
           WRITE LINEA FROM CABECERA1 AFTER PAGE
           WRITE LINEA FROM SUBRAYAR1 AFTER 1
           WRITE LINEA FROM CABECERA2 AFTER 2
           WRITE LINEA FROM SUBRAYAR2 AFTER 1
           MOVE 6 TO CONT-LINEAS.
```



▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼